The ETALIS System Version 1.1 Manual

Draft

Paul Fodor
Darko Anicic
Sebastian Rudolph
Jia Ding
Ahmed Hafsi
Roland Stühmer

September 20, 2010

Contents

Chapter 1

Introduction

Complex Event Processing (CEP) is concerned with timely detection of complex events within multiple streams of atomic occurrences, and has useful applications in areas including financial services, mobile and sensor devices, click stream analysis etc.

ETALIS ([8, 5, 2, 3, 4, 6]) is a research-oriented, commercial-grade Complex Event Processing Logic Programming system for Unix and Windows-based platforms. In addition to providing standard complex event composition operators, ETALIS includes the following features:

- Evaluation of out-of-order events.
- Efficient aggregate functions (sum,min,max,etc.).
- Dynamic insertion and retraction of complex event rules.
- A system of modules.
- A variety of garbage collection techniques.

An *event* represents something that occurs, happens or changes the current state of affairs. For example, an event may signify a problem or an impending problem, a threshold, an opportunity, an information becoming available, a deviation etc. We distinguish between *atomic* and *complex* events. An atomic event is defined as an instantaneous occurrence of interest at a point in time. In order to describe more complex dynamic matters that involve several atomic events, formalisms have been created which allow for combining atomic into *complex events*, using different event operators and temporal relationships. The field of Complex Event Processing has the task of processing streams of atomic events with the goal of detecting complex events according to meaningful event patterns.[1]

We have observed that logic programming can be useful with respect to many concepts of CEP. First, a rule-based formalism (like the one we present in this paper) is expressive enough and convenient to represent diverse complex event patterns. Also declarative rules are free of side-effects (e.g. confluence problem).

[1] Apart from this task (also known as pattern matching), CEP further addresses other issues like event filtering, routing, transformation etc.

Second, integration of query processing, that is essential in many event-based applications, with event processing is easy and natural (e.g. recursive queries). Third, our experience with use of logic programing in implementation of the main constructs in CEP as well as in providing extensibility of a CEP system is very positive and encouraging (e.g. number of code lines in logic programming is significantly smaller than in procedural programming). Ultimately, a logic-based event model allows for *reasoning* over events, their relationships, entire state, and possible contextual knowledge available for a particular domain (application). This feature potentially can enable a new generation of programmers to innovate on novel event-driven applications in AI.

The general task of Complex Event Processing can be described as follows. Within some dynamic setting, events take place. Those *atomic events* are instantaneous, i.e., they happen at one specific point in time and have a duration of zero. Notifications about these occurred events together with their timestamps and possibly further associated data (such as involved entities, numerical parameters of the event, or provenance data) enter the CEP system in the order of their occurrence.

The CEP system further features a set of *complex event descriptions*, by means of which *complex events* can be specified as temporal constellations of atomic events. The complex events thus defined can in turn be used to compose even more complex events and so forth. As opposed to atomic events, those complex events are not considered instantaneous but are endowed with a time *interval* denoting when the event started and when it ended.

The purpose of the CEP system is now to detect complex events within this input stream of atomic events. That is, the system is supposed to notify that the occurrence of a certain complex event has been detected, as soon as the system is notified of an atomic event that completes a sequence which makes up the complex event due to the complex event description. This notification may be accompanied by additional information composed from the atomic events' data. As a consequence of this detection (and depending on the associated data), responding actions can be taken, yet this is outside the scope of this paper.

In summary, the problem we address in our approach is to detect complex events (specified in an appropriate formal language) within a stream of atomic events. Thereby we assume that the timeliness of this detection is crucial and algorithmically optimize our method towards a fast response behavior.

In the next two figures we have the ETALIS and EP-SPARQL diagrams. They are self explanatory and in the Section 5 we also describe the API to access these diferent modules.

1.1 Using This Manual

We adopt some standard notational conventions, such as the name/arity convention for describing events, predicates and functors, + to denote input arguments, − to denote output arguments in database predicates, ? for arguments that may be either input or output and # for arguments that are both input and output (can be changed by the procedure). . Also, the manual uses the UNIX syntax for files and directories except when it specifically addresses other operating systems such as Windows.

Finally, we note that ETALIS is under continuous development, and this document —intended to be the user manual— reflects the current status (Version 1.1) of our system. We take great efforts to create a robust and efficient system, but ETALIS is a research system and is to some degree experimental. While some of

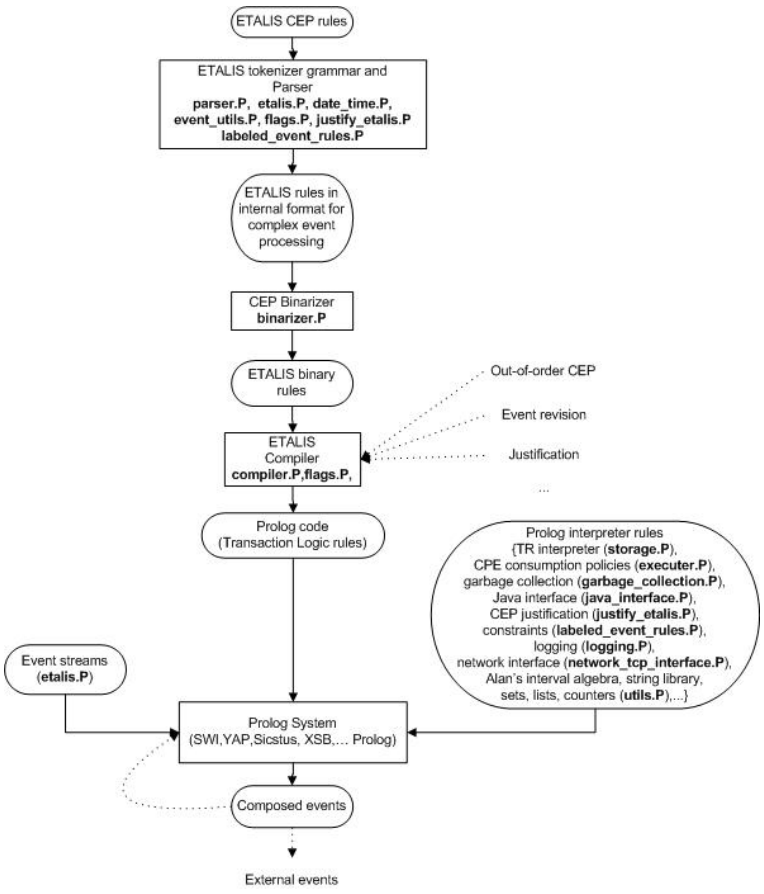

Figure 1.1: The ETALIS system diagram

Version 1.1 is subject to change in future releases, we will try to be as upward-compatible as possible. We would also like to hear from experienced users of our system about features they would like us to include. We do try to accommodate serious users of ETALIS whenever we can.

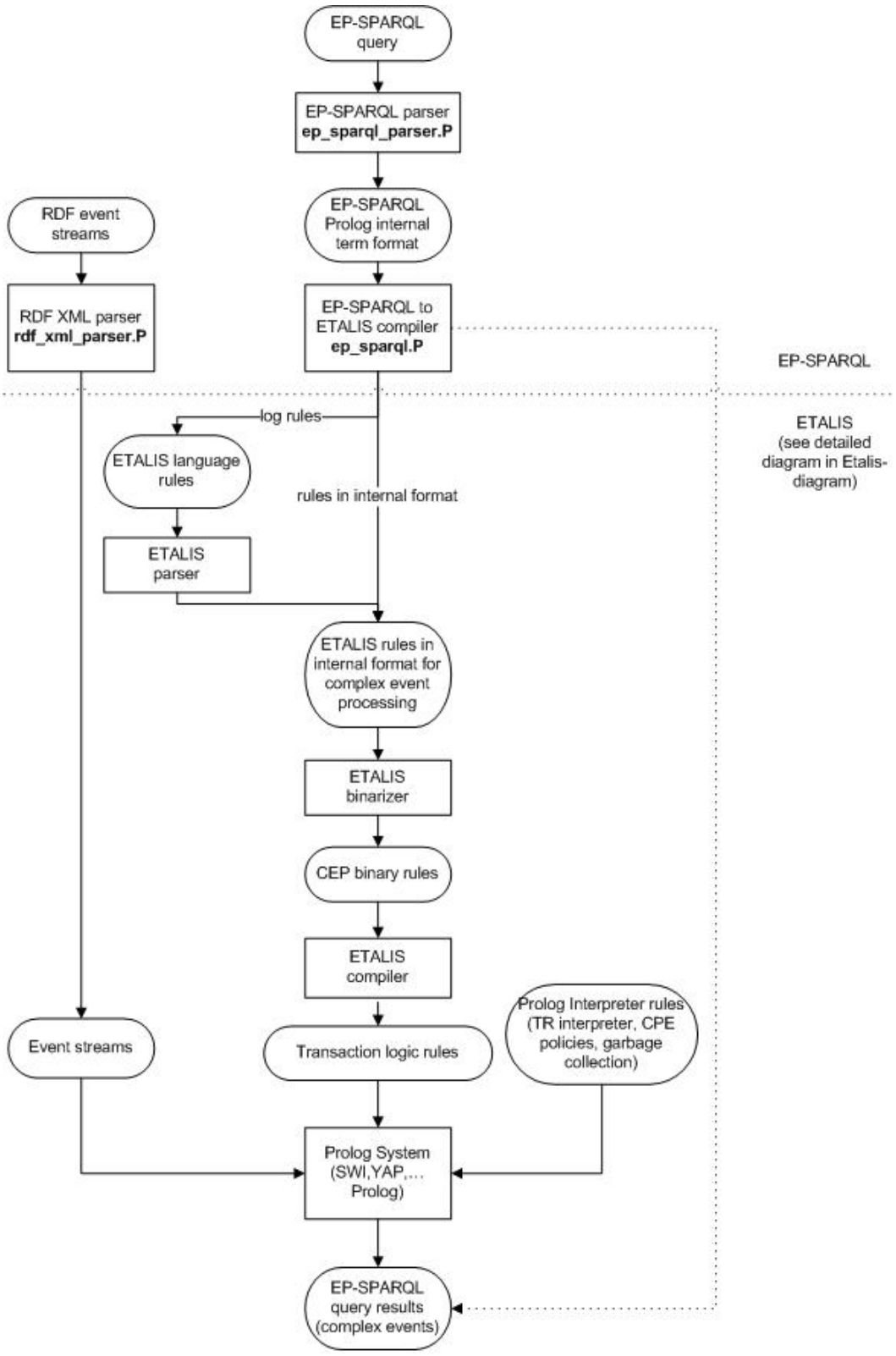

Figure 1.2: The EP-SPARQL diagram

Chapter 2

Getting Started with the ETALIS language

This section describes the steps needed to install ETALIS under UNIX and under Windows.

2.1 Installing ETALIS under UNIX and Windows

The easiest way to install ETALIS is to use the following procedure.

1. Unzip Etalis from the latest release or check out Etalis from the Google code repository:
   http://code.google.com/p/etalis/source/checkout
   using ”subversion”. Linux installations have subversion ”svn” programs, while for Windows, we recommend ”TortoiseSVN” from
   http://tortoisesvn.tigris.org.
   Etalis runs on multiple Prolog systems (we tested SWI, XSB, Yap and Sicstus). However, some features, like the alarm predicates, are supported only under SWI. We will note these exceptions in this document. SWI can be downloaded from:
   http://www.swi-prolog.org/download,
   while specific questions about SWI can be addressed on its mailing list:
   http://www.swi-prolog.org/Mailinglist.txt
   Please use other mailing lists, like the newsgroup comp.lang.prolog, for general Prolog questions. An easy way to access this newsgroup is from the Google group
   http://groups.google.com/group/comp.lang.prolog.

2. Decide in which directory in your file system you want to install ETALIS and copy or move ETALIS there.

2.2 Invoking ETALIS

Under Windows and SWI Prolog, ETALIS can be invoked by the command:

```
$ETALIS_DIR/etalis.bat
```

You will find yourself in the top level Prolog interpreter. You can modify the script for other Prolog systems.

2.3 Compiling ETALIS programs

One way to compile a program from a file, such as `myfile.event` in the current directory and load it into memory, is to type the query:

```
compile_event_file('my_file.event').
```

where `my_file.event` is the name of the file.

2.4 Sample ETALIS programs

There are several sample ETALIS source programs in the directory: `$ETALIS_DIR/examples` illustrating a number of standard features, as well as a number of non-standardized or ETALIS-specific features including operands, garbage collection, windowing, etc.

Hence, a sample session might look like (the actual times shown below may vary and some extra information is given using comments after the % character):

```
my_favourite_prompt> cd $ETALIS_DIR/examples
my_favourite_prompt> test.bat
and_test_01 passed
aggregates_01 passed
alarm_01 passed
cnot_01 passed
during_01 passed
dynamic_updates_01 passed
equals_01 passed
event_multiply_01 passed
flower_delivery_01 passed
...

my_favourite_prompt>
```

2.5 Exiting ETALIS

If you want to exit ETALIS, issue the command `halt.` or simply type `CTRL-d` at the ETALIS prompt. To exit ETALIS while it is executing queries, strike `CTRL-c` a number of times.

Chapter 3

Theoretical basis for the ETALIS language

3.1 The ETALIS language for composing events: syntax and semantics

The syntax of the language allows for the description of *time* and *events*. We represent time instants as well as durations as nonnegative rational numbers $q \in \mathbb{Q}^+$. Events can be atomic or complex. An *atomic event* refers to an instantaneous occurrence of interest. Atomic events are expressed as ground atoms (i.e. predicates followed by arguments which are terms not containing variables). Intuitively, the arguments of a ground atom describing an atomic event denote information items (i.e. event data) that provide additional information about that event.

Atomic events can be composed to form *complex events* via *event patterns*. We use event patterns to describe how events can (or have to) be temporally situated to other events or absolute time points. The language P of event patterns is formally defined by

$$\begin{aligned} P ::= \texttt{pr}(t_1, \ldots, t_n) \quad & \mid P \text{ WHERE } t \mid q \mid (P).q \\ & \mid P \text{ BIN } P \mid \text{NOT}(P, seq(P1, P2)) \end{aligned}$$

Thereby, pr is a predicate name with arity n, t_i denote terms, t is a term of type boolean, q is a nonnegative rational number, and BIN is one of the binary operators SEQ, AND, PAR, OR, EQUALS, MEETS, EQUALS, STARTS, or FINISHES. As a side condition, in every expression p WHERE t, all variables occurring in t must also occur in the pattern p.

Finally, an *event rule* is defined as a formula of the shape

$$\texttt{pr}(t_1, \ldots, t_n) \leftarrow p$$

where p is an event pattern containing all variables occurring in $\texttt{pr}(t_1, \ldots, t_n)$.

Adhering to a stock market scenario, one instantaneous event (not requiring further specification) might be market_closes(). Other events with additional information associated via arguments would be bankrupt$(lehman)$ or buys$(citigroup, wachovia)$. Within patterns, variables instead of constants may occur as arguments, whence we can write bankrupt(X) as a pattern matching all bankruptcy events irrespective of the victim. "Artificial" time-point events can be defined by just providing the according timestamp.

Figure 3.1 demonstrates the various ways of constructing complex event descriptions from simpler ones in the proposed language for event processing. Moreover, the figure informally introduces the semantics

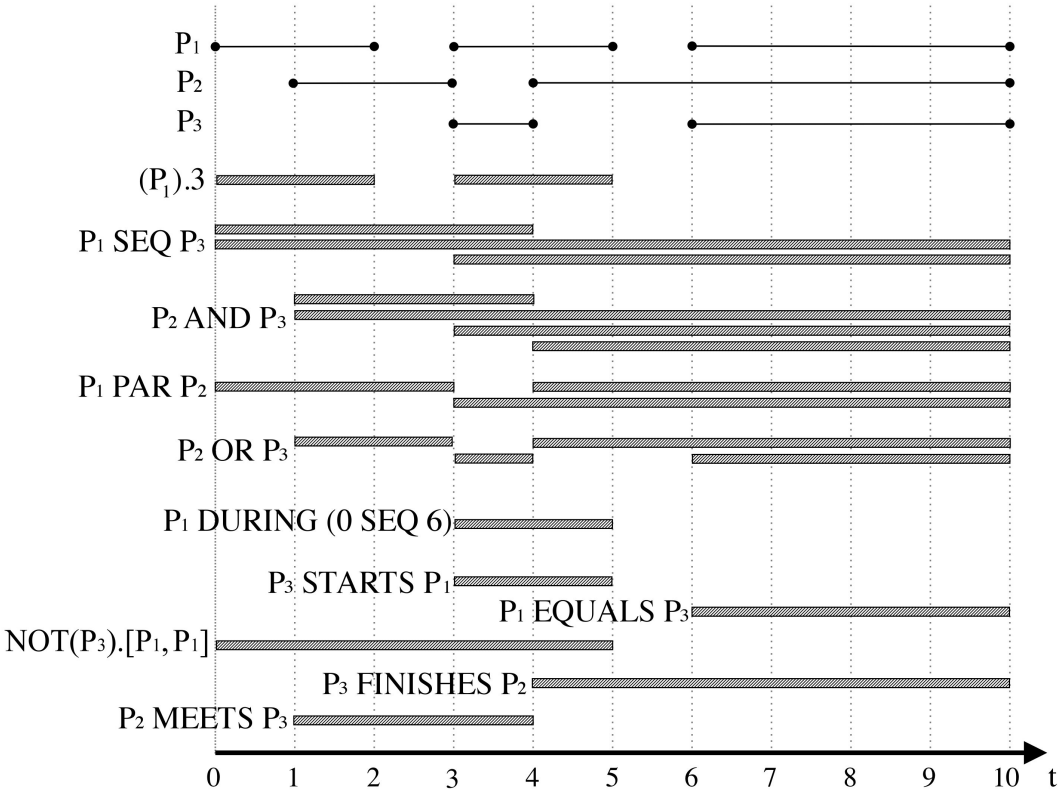

Figure 3.1: Language for Events Processing-Composition Operators

of the language. Let us assume that instances of three complex events, P_1, P_2, P_3, are occurring in time intervals as shown in Figure 3.1. Vertical dashed lines depict different time units, while the horizontal bars represent detected complex events for the given patterns.

It is worth noting that the defined pattern language captures the set of all possible 13 relations on two temporal intervals as defined in [1]. The set can also be used for rich temporal reasoning.

In the following examples, event patterns are considered under the *unrestricted policy*. In event processing, consumption policies deal with an issue of *selecting* particular events occurrences when there are more than one event instance applicable and *consuming* events after they have been used in patterns.

It is worthwhile to briefly review the modeling capabilities of the presented pattern language. For example, one might be interested in defining an event matching stock market working days:

$\texttt{workingDay}() \leftarrow \text{NOT}(\texttt{marketCloses}())[\texttt{marketOpens}(), \texttt{marketCloses}()].$

Moreover, we might be interested in detecting the event of two bankruptcies happening on the same market working day:

$\texttt{dieTogether}(X, Y) \leftarrow$
$\big(\texttt{bankrupt}(X) \text{ SEQ } \texttt{bankrupt}(Y)\big) \text{ DURING } \texttt{workingDay}().$

This event rule also shows, how event information (about involved institutions, provenance, etc.) can be "passed" on to the defined complex events by using variables. Furthermore, variables may be employed to conditionally group events into complex ones if they refer to the same entity:

$\texttt{indirectlyAcquires}(X, Y) \leftarrow \texttt{buys}(Z, Y)$ AND $\texttt{buys}(X, Z)$

Even more elaborate constraints can be put on the applicability of a pattern by endowing it with a boolean type term as filter.[1] Thereby, we can detect a stock prize increase of at least 50% in a time frame of 7 days.

$\texttt{remarkableIncrease}(X) \leftarrow$
$(\texttt{prize}(X, Y_1)$ SEQ $\texttt{prize}(X, Y_2)).7$ WHERE $Y_2 > Y_1 \cdot 1.5$

This small selection arguably demonstrates the expressivity and versatility of the introduced language.

We define the declarative formal semantics of the proposed language for event processing in a model-theoretic way.

Note that we assume a fixed interpretation of the occurring function symbols, i.e. for every function symbol f of arity n, we presume a predefined function $f^* : Con^n \to Con$. That is, in our setting, functions are treated as built-in utilities.

As usual, a *variable assignment* is a mapping $\mu : Var \to Con$ assigning a value to every variable. We let μ^* denote the extension of μ to terms defined in the usual way:

$$\mu^* : \begin{cases} v & \mapsto \mu(v) & \text{if } v \in Var, \\ c & \mapsto c & \text{if } c \in Con, \\ f(t_1, \ldots, t_n) & \mapsto f^*(\mu^*(t_1), \ldots, \mu^*(t_n)) & \text{otherwise.} \end{cases}$$

In addition to the set of rules $\mathcal{R}$, we fix an *event stream*. The event stream is formalized as a mapping $\epsilon : Ground \to 2^{\mathbb{Q}^+}$ from ground predicates into sets of nonnegative rational numbers. It thereby indicates at what time instants what elementary events occur. As a side condition, we require ϵ to be free of accumulation points, i.e. for every $q \in \mathbb{Q}^+$, the set $\{q' \in \mathbb{Q}^+ \mid q' < q \text{ and } q' \in \epsilon(g) \text{ for some } g \in Ground\}$ is finite.

Now, we define an interpretation $\mathcal{I} : Ground \to 2^{\mathbb{Q}^+ \times \mathbb{Q}^+}$ as a mapping from the ground atoms to sets of pairs of nonnegative rationals, such that $q_1 \leq q_2$ for every $\langle q_1, q_2 \rangle \in \mathcal{I}(g)$ for all $g \in Ground$.

Given an event stream ϵ, an interpretation $\mathcal{I}$ is called a *model* for a rule set $\mathcal{R}$ – written as $\mathcal{I} \models_\epsilon \mathcal{R}$ – if the following conditions are satisfied:

C1 $\langle q, q \rangle \in \mathcal{I}(g)$ for every $q \in \mathbb{Q}^+$ and $g \in Ground$ with $q \in \epsilon(g)$

C2 for every rule $atom \leftarrow pattern$ and every variable assignment μ we have $\mathcal{I}_\mu(atom) \subseteq \mathcal{I}_\mu(pattern)$ where $\mathcal{I}_\mu$ is inductively defined as displayed in Fig. 3.2.

Given an interpretation $\mathcal{I}$ and some $q \in \mathbb{Q}^+$, we let $\mathcal{I}|_q$ denote the interpretation defined via $\mathcal{I}|_q(g) = \mathcal{I}(g) \cap \{\langle q1, q2 \rangle \mid q2 - q1 \leq q\}$.

Given two interpretations $\mathcal{I}$ and $\mathcal{J}$, we say that $\mathcal{I}$ is *preferred* to $\mathcal{J}$ if there exists a $q \in \mathbb{Q}^+$ with $\mathcal{I}|_q \subset \mathcal{J}|_q$.

A model $\mathcal{I}$ is called *minimal* if there is no other model preferred to $\mathcal{I}$. It is easy to show that for every event stream ϵ and rule base $\mathcal{R}$ there is a unique minimal model $\mathcal{I}^{\epsilon,\mathcal{R}}$.

[1] Note that also comparison operators like $=, <$ and $>$ can be seen as boolean-typed binary functions and, hence, fit well into the framework.

pattern	$\mathcal{I}_\mu$(pattern)
$\mathtt{pr}(t_1,\ldots,t_n)$	$\mathcal{I}(\mathtt{pr}(\mu^*(t_1),\ldots,\mu^*(t_n)))$
p WHERE t	$\mathcal{I}_\mu(p)$ if $\mu^*(t) = \mathit{true}$
	$\emptyset$ otherwise.
q	$\{\langle q,q\rangle\}$ for all $q \in \mathbb{Q}^+$
$(p).q$	$\mathcal{I}_\mu(p) \cap \{\langle q_1,q_2\rangle \mid q_2 - q_1 = q\}$
p_1 SEQ p_2	$\{\langle q_1,q_4\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2,q_3 \in \mathbb{Q}^+$ with $q_2 < q_3\}$
p_1 AND p_2	$\{\langle \min(q_1,q_3), \max(q_2,q_4)\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2,q_3 \in \mathbb{Q}^+\}$
p_1 PAR p_2	$\{\langle \min(q_1,q_3), \max(q_2,q_4)\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2,q_3 \in \mathbb{Q}^+$ with $\max(q_1,q_3) < \min(q_2,q_4)\}$
p_1 OR p_2	$\mathcal{I}_\mu(p_1) \cup \mathcal{I}_\mu(p_2)$
p_1 EQUALS p_2	$\mathcal{I}_\mu(p_1) \cap \mathcal{I}_\mu(p_2)$
p_1 MEETS p_2	$\{\langle q_1,q_3\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_2,q_3\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2 \in \mathbb{Q}^+\}$
p_1 DURING p_2	$\{\langle q_3,q_4\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_3,q_4\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2,q_3 \in \mathbb{Q}^+$ with $q_3 < q_1 < q_2 < q_4\}$
p_1 STARTS p_2	$\{\langle q_1,q_3\rangle \mid \langle q_1,q_2\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_1,q_3\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2 \in \mathbb{Q}^+$ with $q_2 < q_3\}$
p_1 FINISHES p_2	$\{\langle q_1,q_3\rangle \mid \langle q_2,q_3\rangle \in \mathcal{I}_\mu(p_1)$ and $\langle q_1,q_3\rangle \in \mathcal{I}_\mu(p_2)$ for some $q_2 \in \mathbb{Q}^+$ with $q_1 < q_2\}$
NOT$(p_1).[p_2,p_3]$	$\mathcal{I}_\mu(p_2$ SEQ $p_3) \setminus \mathcal{I}_\mu(p_2$ SEQ p_1 SEQ $p_3)$

Figure 3.2: Definition of extensional interpretation of event patterns. We use $p_{(x)}$ for patterns, $q_{(x)}$ for rational numbers, $t_{(x)}$ for terms and pr for predicates.

Finally, given an atom a and two rational numbers q_1, q_2, we say that the event $a^{[q_1,q_2]}$ is a *consequence* of the event stream ϵ and the rule base $\mathcal{R}$ (written $\epsilon, \mathcal{R} \models a^{[q_1,q_2]}$), if $\langle q_1,q_2\rangle \in \mathcal{I}_\mu^{\epsilon,\mathcal{R}}(a)$ for some variable assignment μ.

It can be easily verified that the behavior of the event stream ϵ beyond the time point q_2 is irrelevant for determining whether $\epsilon, \mathcal{R} \models a^{[q_1,q_2]}$ is the case.[2] This justifies to take the perspective of ϵ being only partially known (and continuously unveiled along a time line) while the task is to detect event-consequences as soon as possible.

3.2 Event processing execution in Prolog

The syntax of *ETALIS Language for Events* allows for the description of *event* patterns as event rules of the form: $complexEvent \leftarrow EventPattern$. Events occur over time intervals. Time instants as well as durations are modeled as nonnegative rational numbers $q \in \mathbb{Q}^+$. Events can be atomic or complex, while no distinction is made in their applicability to rules. An *atomic event* refers to an instantaneous occurrence, i.e., the time interval length is zero. Althought not a requirement, atomic events are ground (i.e. predicates followed by arguments which are terms not containing variables). Intuitively, the arguments of a ground atom describing an atomic event denote information items (i.e. event data) that provide additional

[2]More formally, for any two event streams ϵ_1 and ϵ_2 with $\epsilon_1(g) \cap \{\langle q,q'\rangle \mid q' \leq q_2\} = \epsilon_2(g) \cap \{\langle q,q'\rangle \mid q' \leq q_2\}$ we have that $\epsilon_1, \mathcal{R} \models a^{[q_1,q_2]}$ exactly if $\epsilon_2, \mathcal{R} \models a^{[q_1,q_2]}$.

information about the event.

Events participate in composition rules to trigger complex events. When an *event stream* of atomic events is fed into the system, all patterns are considered and complex events are triggered. The event stream is formalized as a mapping $\epsilon : Ground \rightarrow 2^{\mathbb{Q}^+}$ from ground predicates into sets of nonnegative rational numbers. It thereby indicates at what time instants what simple events occur. As a side condition, it is required that ϵ is free of accumulation points, i.e. for every $q \in \mathbb{Q}^+$, the set $\{q' \in \mathbb{Q}^+ \mid q' < q \text{ and } q' \in \epsilon(g) \text{ for some } g \in Ground\}$ is finite.

Given an event stream ϵ, an interpretation $\mathcal{I}$ is called a *model* for a rule set $\mathcal{R}$ – written as $\mathcal{I} \models_\epsilon \mathcal{R}$ – if the following conditions are satisfied:

Given a set of event patterns and a stream of input events, the ETALIS system can compute the final model of all events. To achieve this, ETALIS implements *event-driven backward chaining* rules. These rules are executed in a *data-driven* fashion. That is the inference system incrementally furthers the pattern completion as relevant events occur. As soon as the last event required for a pattern fulfillment is observed, the inference system triggers the complex event.

A user defines event patterns of the form given in the left column of Figure 3.2. When submitted, ETALIS automatically transforms these patterns into event-driven backward chaining rules. These are executable rules that enable detection of complex events at run time. The transformation is sketched as follows.

First, an event pattern is binarized left associatively, i.e., operations are coupled to generate only binary formulas, introduce intermediate events for every binary formula and replace these formulas in the original program. This eases the process of automatic construction of *event-driven* rules and helps in implementation of various event operators defined by the language semantics (Figure 3.2). Apart from this, the consideration of events on "two by two" basis enhances the computation sharing in the pattern detection, and hence helps in achieving better run-time performance. For instance, a formula: $e \leftarrow p_1$ SEQ p_2 SEQ p_3... SEQ p_n (e is detected when an event p_1 is followed by p_2,.., followed by p_n) is binarized by introducing intermediate events (goals) as:

$$\begin{aligned} &e \leftarrow temp_{n-1} \text{ SEQ } p_n \\ &temp_{n-1} \leftarrow temp_{n-2} \text{ SEQ } p_{n-1} \\ &... \\ &temp_1 \leftarrow p_1 \text{ SEQ } p_2 \end{aligned} \tag{3.1}$$

Second, each binary formula is then compiled into a set of event-driven backward chaining rules (i.e., executable rules). Each operator, defined by the language semantics, has a specific transformation which is provided by ETALIS system. Due to the space restriction, only the transformation for the sequential conjunction is sketched below. Implementation of other operators follow similar design patterns.

The transformation accepts as input a binary sequence $e_i \leftarrow a$ SEQ b, and produces event-driven backward chaining rules[3]. These rules are represented by $r(a)_1$ and $r(b)_1$ in Transformation 2.1. They belong to two different classes of rules. We refer to the first class as to rules used to *generate goals*. The second class correspond to *checking rules*. $r(a)_1$ is a rule that generates goals of type $goal(b^{[\text{-,-}]}, a^{[T_1,T_2]}, e_i^{[\text{-,-}]})$ when an

[3]Here we assume that the process of binarization (which is trivial) has already been completed so that Transformation 2.1 accepts as input only binary patterns.

Sequential conjunction.

Input: event binary goal $e_i \leftarrow a$ SEQ b.

Output: event-driven backward chaining rules for SEQ operator.

For each event binary goal $e_i \leftarrow a$ SEQ b {

whenever a occurs at some $[T_1, T_2]$, apply all rules $r(a)_i$:

$r(a)_1$:- insert $goal(b^{[\_,\_]}, a^{[T_1,T_2]}, e_i^{[\_,\_]})$;

whenever b occurs at some $[T_3, T_4]$, apply all rules $r(b)_j$:

$r(b)_1$:- **if** $goal(b^{[\_,\_]}, a^{[T_1,T_2]}, e_i^{[\_,\_]})$ exist and $T_2 < T_3$ **then**

delete that goal, and trigger event $e_i^{[T_1,T_4]}$;

end if

}

event a occurs (i.e., when the rule head $r(a)_1$ is satisfied) at some $[T_1, T_2]$. Its interpretation is that "an event a has occurred at $[T_1, T_2]$[4], and we are waiting for b to happen, in order to detect e_i". Obviously the goal does not carry information about times for b and e_i, as we don't know when they will occur. In general, the *second* event in a goal always denotes an event that has just occurred, whereas the role of the *first* event is to specify what we are waiting for, to detect an event that is on the *third* position. Now when an event b happens at some $[T_3, T_4]$, the rule $r(b)_1$ will execute. The rule checks whether $goal(b^{[\_,\_]}, a^{[T_1,T_2]}, e_i^{[\_,\_]})$ is true (meaning that an a occurred prior to the occurrence of b, if $T_2 < T_3$) in which case it triggers a (more) complex event $e_i^{[T_1,T_4]}$. Additionally the rule deletes $goal(b^{[\_,\_]}, a^{[T_1,T_2]}, e_i^{[\_,\_]})$ to free up the memory (this is an optional operation, and in certain applications it may be omitted).

Another important issue in event processing is to consider different consumption policies when detecting complex events. Let us assume that we want to detect event a followed by event b, and the stream produces events: a, a, b. It is a question which event a will be taken for the pattern detection, the first or the second instance. In event processing, consumption policies (or event contexts) deal with an issue of selecting particular events occurrences when there are more than one event instance applicable and consuming events after they have been used in patterns. In ETALIS we have implemented *recent*, *chronological*, and *unrestricted* policy; and for practical use with out-of-order events, recent and chronological policies are used.

3.3 Out-of-order event detection in ETALIS

To explain our approach which deals with late events let us consider a simple event binary goal: $e_i \leftarrow a$ SEQ b (using the binarization, other more complicated examples can also be reduced to this case). The solution slightly modifies the initial Transformation 2.1. by adding additional rules. A rule that generates a goal (i.e., $r(a)_1$) is accompanied by a checking rule (i.e., $r(a)_2$) and vise versa (the checking rule, $r(b)_1$, is now added a rule that generates a goal, $r(b)_2$, see also Section **??**). Therefore we process the sequence in both directions: an in-order direction (as in Transformation 3.2); and an out-of-order direction (with newly added rules in Transformation 3.1.). Although, we show here just the transformation for the sequence operator, we have implemented transformations for all thirteen operators inspired from Allen's Interval Algebra and also our additional various operators dealing with negation, constraints on event rules and

[4] Apart from the time stamp, an event may carry other data parameters that are omitted here in order to make rules more readable.

aggregates.

Rules $r(a)_1$ and $r(a)_2$ will be evaluated when an event $a^{[T_1,T_2]}$ occurs (i.e., at $[T_1, T_2]$). Rule $r(a)_1$ will insert a goal $goal(b^{[\_,\_]}, a^{[T_1,T_2]}, e_i^{[\_,\_]})$ into the database. Additionally rule $r(a)_2$ will check whether the event a is an out-of-order event, in which case the system will also trigger an event e_i. The event a is an out-of-order event if a goal $goal\_out(a^{[\_,\_]}, b^{[T_3,T_4]}, e_i^{[\_,\_]})$ exists in the database, and $T_2 < T_3$. The latter condition says that although event $a^{[T_1,T_2]}$ just happened (at some $[T_1, T_2]$), there is an event $b^{[T_3,T_4]}$ that has already happened such that its timestamp is bigger that the a's timestamp. This suggests that event a is an out-of-order event, and an event $e_i^{[T_1,T_4]}$ should be indeed triggered.

Rules, that will fire when an event $b^{[T_3,T_4]}$ occurs (at some $[T_3, T_4]$), work similarly as those for $a^{[T_1,T_2]}$. Rule $r(b)_1$ will check whether an event $a^{[T_1,T_2]}$ has already happened (i.e., $goal(b^{[\_,\_]}, a^{[T_1,T_2]}, e_i^{[\_,\_]})$ exists in the database); and if yes, it will trigger an event $e_i^{[T_1,T_4]}$. That is an in-order case of processing events a and b. Additionally rule $r(b)_2$ will insert a goal $goal\_out(a^{[\_,\_]}, b^{[T_3,T_4]}, e_i^{[\_,\_]})$, which will be used by $r(a)_2$ if an out-of-order event a occurs.

Sequence with Out-of-Order Events. **Input:** event binary goal $e_i \leftarrow a$ SEQ b.

Output: event-driven backward chaining rules for SEQ operator.

For each event binary goal $e_i \leftarrow a$ SEQ b {

whenever $a^{[T_1,T_2]}$ occurs apply all rules $r(a)\_i$:

$r(a)_1$:- insert $goal(b^{[\_,\_]}, a^{[T_1,T_2]}, e_i^{[\_,\_]})$;

$r(a)_2$:- **if** $goal\_out(a^{[\_,\_]}, b^{[T_3,T_4]}, e_i^{[\_,\_]})$ exist and $T_2 < T_3$ **then**

delete that goal and trigger event $e_i^{[T_1,T_4]}$;

end if

whenever $b^{[T_3,T_4]}$ occurs apply all rules $r(b)\_j$:

$r(b)_1$:- **if** $goal(b^{[\_,\_]}, a^{[T_1,T_2]}, e_i^{[\_,\_]})$ exist and $T_2 < T_3$ **then**

delete that goal and trigger event $e_i^{[T_1,T_4]}$;

end if

$r(b)_2$:- insert $goal\_out(a^{[\_,\_]}, b^{[T_3,T_4]}, e_i^{[\_,\_]})$;

}

Effectively, the price paid for handling out-of-order events is mainly reflected throughout insertion of out-of-order goals (e.g., $goal\_out(a^{[\_,\_]}, b^{[T_3,T_4]}, e_i^{[\_,\_]})$) and the fact that they need to be cleared up after certain time (to free up the memory). Therefore, in the next section we discuss a solution for the effective garbage collection of outdated out-of-order goals.

3.4 Memory management

To deal with out-of-order events safely, no data can ever be purged from memory since event processing assumes processing of infinite streams of data. However, this requirement is an exaggeration in reality and is impracticable due to overuse of memory. Network latencies can be approximated, so it is clear that, at some point, data must be deleted from memory. In the transformation above, occurrences of each event are recorded by inserting a goal in memory. Some of these goals are removed at the time they are "consumed" to

build more complex events, while the others can be pruned using a time window[5]. Due to the requirement in CEP that patterns are defined on time windows, we have developed time-based garbage collection strategies and not triggered by the memory consumption ones like in many other fields. The time-based garbage collection is the natural approach for CEP to release the memory necessary for the execution of events.

We have implemented the time guarantees for out-of-order event detection in different ways: pushed constraints; general garbage collection; and event-pattern garbage collection.

The common way to deal with garbage collection of overdue events is to define a time window for the event pattern and check this constraint during the composition of the complex event. For instance, an event binary goal:
$ruleId([ooo\_window(10)])rule: \quad e_i \leftarrow a$ SEQ b SEQ c.
specifies that the length of a time window for out-of-order events is 10 seconds (i.e., $ooo\_window(10)$). This means the system guarantees that out-of-order events will be processed correctly if their delay is shorter that the specified window.

3.4.1 Pushed constraints

Our first implementation for out-of-order complex event detection in ETALIS modifies the binarization by pushing the constraints for time guarantees into binary events during binarization, and Transformation 3.1 with checking the constraints before triggering composed events. Pushing the constraints during binarization ensures that time guarantees are checked at each step, so unnecessary intermediary sub-complex events are not generated if the time guarantees are not satisfied. For predicative rules, we push variants for all the terms and variables used in the rule to ensure that all bindings are satisfied during execution (equivalent to a lifting from propositional to predicative logic).

Sequence with constraint checks.

Input: event binary goal $RuleLabelConditionse_i \leftarrow a$ SEQ b.

Output: event-driven backward chaining rules for SEQ operator.

For each event binary goal $RuleLabelConditionse_i \leftarrow a$ SEQ b {

whenever $a^{[T_1,T_2]}$ occurs apply all rules $r(a)_i$:

$r(a)_1$:- insert $goal(b^{[-,-]}, a^{[T_1,T_2]}, e_i^{[-,-]})$;

$r(a)_2$:- **if** $goal\_out(a^{[-,-]}, b^{[T_3,T_4]}, e_i^{[-,-]})$ exist and $T_2 < T_3$
and $check\_constraints(RuleLabelConditions)$ **then**
delete that goal and trigger event $e_i^{[T_1,T_4]}$;
end if

whenever $b^{[T_3,T_4]}$ occurs apply all rules $r(b)_j$:

$r(b)_1$:- **if** $goal(b^{[-,-]}, a^{[T_1,T_2]}, e_i^{[-,-]})$ exist and $T_2 < T_3$
and $check\_constraints(RuleLabelConditions)$ **then**
delete that goal and trigger event $e_i^{[T_1,T_4]}$;
end if

$r(b)_2$:- insert $goal\_out(a^{[-,-]}, b^{[T_3,T_4]}, e_i^{[-,-]})$;

}

---

[5]When specified time elapses, goals from unfulfilled patterns can be deleted.

One advantage of this approach is that any constraints can be verified, not only for out-of-order event detection. Such constraints are common in event processing, e.g., the event detection started after or before a certain time. Moreover, this approach is declarative, i.e., new constraints can be defined for any rule and the handling of the constraints is defined by writing a user defined $check\_constraint$ rule for that constraint type. However, the approach also has important disadvantages. First, ETALIS enables sharing of common formulas during binarization (i.e., shared intermediate complex events are computed only once and shared in multiple event formulas). Pushing the constraints and labels for each rule makes sharing not possible anymore. However, a bigger disadvantage is the fact that the time guarantee is checked for each detected event. An efficient solution would clear events when they are overdue, i.e., not every time an event is detected. For instance, if the system detects 100,000 events in two seconds and the time window is set to 2 seconds, then the system is expected to clean the overdue events only once (after two seconds), i.e., without performing 100,000 checks.

3.4.2 General and pattern-based garbage collection

We prune expired goals periodically using alarm predicates. The general approach for garbage collection (GC) is utilized to reduce an event path on which out-of-order events are processed (similarly as in [7]). Essentially it enables an out-of-order event to be late for a fixed window of time with respect to system clock, denoted by $SystemClock$. The GC window W specifies the maximum time range between the first and last event for any pattern detection (i.e., infinitely long complex patterns are of no interest). Every event $e_i^{[T_1,T_2]}$ should be kept in memory at least the time defined by W, and all events are allowed to be purged if $SystemClock > [T_1 + W]$. GC is applied for all intermediate goals, not only for out-of-order event processing.

We use an alarm rule (3.2) to prune unnecessary goals. This, sort of, garbage collector is triggered by the system generated events (defined by the system time $SystemClock$ and the GC window W).

$$
\begin{array}{l}
garbageCollector(SystemClock) \leftarrow \\
\quad findAll(goal(\_, X([T_1, T_2], W), \_) \text{ SEQ } SystemClock > [T_1 + W], \\
\quad goal(\_, X([T_1, T_2]), \_, L)), \\
\quad\quad while\_do(member(goal(\_, X([T_1, T_2]), \_, L)))(\\
\quad\quad del(goal(\_, X([T_1, T_2]), \_))) \\
\quad and\ alarm(garbageCollector(SystemClock + W), W).
\end{array}
\tag{3.2}
$$

This means that for a time window of 10 seconds, the following sequence of events will not be detected by the rule (**??**): $stock(agent1,'' G'', 110, 10)$, that is triggered and received at time 2; and $stock(agent1,'' G'', 100, 10)$, that is triggered at time 1 and received at time 21. The general garbage collection works well when there is a single garbage collection window W for the whole system (e.g., the network delay is the same for all sources).

The window essentially specifies what is a guaranteed "minimum" time, ensured by the system, that out-of-order events will be processed correctly: if the GC via alarms is set to W time window, the presented procedure correctly handles out-of-order events within the window.

Let us consider now a case when different elements in the system have different delays and time guarantees, i.e., there exist different garbage collection times for different patterns. In this case, the garbage collection alarms are defined at the level of each rule. The procedure starts GC alarms for each rule separately,

looking for intermediate goals for those rules checking the condition $SystemClock > [T_1 + Window(e_i)]$.

Similarly to the pushed constraints case, rules are defined with properties, and the binarization pushes the rule properties to sub-components. However, alarm events for garbage collection are scheduled to happen in $Window(e_i)$ time. The scheduling of alarms is done right after the compilation of pattern rules in an event program. The approach is conservative: if one writes patterns without garbage collection window, no alarm is generated. However, we also permit dynamic properties by inserting/deleting properties on-the-fly $ins/del(property(RuleId, PropertyName, PropertyValue))$. In this case, the GC is started automatically during the execution (depending on the situation). The model theory is also extended with a augmentation theory for handling rule properties, i.e., we have normal labeled CEP rules and and special predicates defining features of rules can be considered as the the meta-theory over the event logic $property(RuleId, PropertyName, PropertyValue)$. This makes the system more extensible: by adding new properties or meta-theories, we do not modify the previous semantic framework.

3.5 Justification

Debugging is one of the determining issues in the longevity of using any programming language. Efficient debugging tools are needed by every programmer; and event-oriented programming is not an exception.

Justification is done by adding *justification edges* in a justification-deduction graph. When event b occurs at some $[T_3, T_4]$, the third rule will insert $jstf\_edge$. The edge establishes a causal relationship saying that c happened as a consequence of a sequence "a folloed by b". The edge will be inserted only if event a indeed happened prior to event b. This condition is ensured by checking whether the goal $goal(\mathtt{b}, \mathtt{a}, \mathtt{c})$ and the condition, $T_2 < T_3$, are both true (see the third rule); and these will be true if event a happened prior to an occurrence of event b (i.e., the second rule inserts goal $goal(\mathtt{b}, \mathtt{a}, \mathtt{c})$ when event a happens).

$$
\begin{array}{rl}
\mathtt{c} \leftarrow & \mathtt{a}\ \text{SEQ}\ \mathtt{b}. \\
\mathtt{a}[T_1, T_2] \leftarrow & insert(goal(\mathtt{b}[\_, \_], \mathtt{a}[T_1, T_2], \mathtt{c}[\_, \_])). \\
\mathtt{b}[T_3, T_4] \leftarrow & goal(\mathtt{b}[T_3, T_4], \mathtt{a}[T_1, T_2], \mathtt{c}[\_, \_]), T_2 < T_3, \\
& \mathtt{c}[T_1, T_4], insert(\\
& jstf\_edge(\\
& \quad \mathtt{event}(\mathtt{c}, [T_1, T_4]), \\
& \quad \mathtt{clause\_body}(\\
& \quad\quad Lbl, \mathtt{seq}(\mathtt{event}(a, [T_1, T_2]), \mathtt{event}(b, [T_3, T_4])) \\
&))).
\end{array}
$$

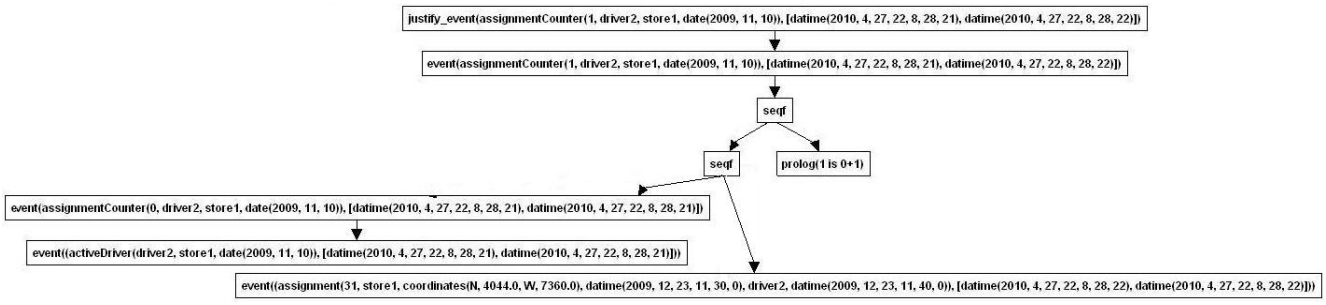

Figure 3.3: Justification of a successful occurrence of event $assignmentCounter$

When a user wants to get a justification for a certain event, the following rule will be evaluated.

$$
\begin{array}{l}
\texttt{jstf\_ev}(Ev, [T_1, T_2], J) \leftarrow \\
\quad \texttt{jstf\_edge}(\\
\quad\quad \texttt{event}(Ev, [T_1, T_2]), \\
\quad\quad \texttt{clause\_body}(\\
\quad\quad\quad Lbl, \texttt{seq}(\texttt{event}(I_1, [T_1', T_2']), \texttt{event}(I_2, [T_3', T_4'])) \\
\quad)) \\
\quad \texttt{jstf\_ev}(I_1, [T_1', T_2'], J_2), \\
\quad \texttt{jstf\_ev}(I_2, [T_3', T_4'], J_3), \\
\quad J = \texttt{jstf}(Lbl, \texttt{head}(Ev, [T_1, T_2]), \texttt{seq}(J_2, J_3)).
\end{array}
$$

The rule effectively traverses the justification-deduction graph (that consists of complex events split up into binary events). The graph is traversed top-down starting from the complex event (that needs to be justified). The rule is recursive; it calls itself until all pairs of events that build the complex event are found. Due to space restriction we have shown only the justification rule for the sequence operator.

Chapter 4

Interacting with the ETALIS system

Throughout this chapter, we use `$ETALIS\_DIR` to refer to the directory in which ETALIS was installed.

4.1 Entering and exiting ETALIS from the command Line

After the system has been installed, the emulator's executable code appears in the file:

```
$ETALIS\_DIR/etalis.bat
```

or it can be loaded in the Prolog system with:

```
['$ETALIS\_DIR/src/etalis.P']
```

Using the `-g` command-line option in Yap or SWI Prolog any goal can be can be executed, up to 1024 characters. For instance,

```
$ETALIS\_DIR/bin/ETALIS -g ``compile_event_file('hello-world.event'),
       execute\_event\_stream\_file('-stream.P'),halt.''
```

loads and event file "hello-world.event", executes all the events from a file "stream.P" and exits ETALIS. Within the 1024 character limit, any query or command can be executed.

There are several ways to exit ETALIS. A user may issue the command `halt.` or `end\_of\_file.`, or simply type `CTRL-d` at the ETALIS prompt. To interrupt ETALIS while it is executing a query, strike `CTRL-c`.

4.2 The system and its directories

When installed, the ETALIS system resides in a single directory that contains several subdirectories.

1. `docs` contains the user manuals and other documentation, including the technical documentation manual for developers.
2. `examples` contains some examples for all features of ETALIS.
3. `lib` contains links to external libraries used in ETALIS.
4. `src` The directory `src` contains the sources of ETALIS. These files are written in Prolog and are loaded in the main file "etalis.P".
5. `www` contains the WWW interface to ETALIS, which is written in Java Server Pages (JSP).

4.3 The module system of ETALIS

ETALIS has been designed with a basic module system in mind. Modules provide a small step towards *logic programming "in the large"* that facilitates the construction of large programs or projects from components that are developed separately and support the principle of information hiding.

Altough, only partially implemented, the Etalis module is similar to that of Prolog systems and is *flat*, that is modules cannot be nested. We also allow creation of modules dynamically and updates of data in all modules, including deletion of modules. Files are not treated as modules, but are loaded in modules.

`:- export` sym_1`, ...,` sym_l`.`

`:- import` sym_1`, ...,` sym_n `from` $file$`.`

where sym_i has the form $functor/arity$, and $module$ is a Prolog atom representing a module name.

In ETALIS, the declaration

`:- export` sym_1`, ...,` sym_l`.`

specifies that the current file exports only a certain set of event definitions.

The declaration

`:- import` sym_1`, ...,` sym_n `from` $file$`.`

allows events to be imported from a file. These predicates can be used in the main user space without any prefix.

All the other event and `e/n` predicate symbols `p/n` are identified as if they were prefixed with their module name (i.e. base file name). Hence the occurrence of `e/n` in two different modules, `m1` and `m2` are distinct symbols that can be denoted as `m1:e/n` and `m2:e/n`.

4.4 Compiling and loading event files

ETALIS provides for both statically compiled code and dynamically asserted code. The standard predicate `compile_even_file/1` is the most convenient method for entering static source code rules into ETALIS's database. Compiling a file `File` consists of the following steps.

Name Resolution: determine the file that `File` designates.

Parsing: parsing the event file into internal rule representation.

Binarization: binarize the event rules. The result of this step is a set of binary rules where the bodies consist of at most 2 events and one ETALIS operator.

Compilation of event rules: compile the file into a format close to Prolog and Transaction Logic.

Loading: load the resulting program into memory.

Event Stream Execution: execute an event stream detecting the complex events. The events are considered in the order in which they are provided by the consumption policy module.

```
| ?- compile_event_file(Files).
```

For a given, `File` to be compiled, the source file name corresponding to `File` is obtained by concatenating a directory prefix and the extension `.event`. The directory prefix must be in the dynamic loader path.

4.5 ETALIS options and flags

The ETALIS system has several execution options.

1. `etalis_justification/1:` enables the justification. Can take the values "on" and "off". It is defined in the file "compiler.P".
2. `event_consumption_policy/1:` sets the event consumtpion policy. Can be: "recent", "chronological" and "unrestricted" (defined in "event_utils.P"). The default setting is "event_consumption_policy(recent)".
3. `garbage_control/1:` enables various types of garbage collection, that is goals older than a certain period of time are cleaned. Can have the the values defined in "garbage_collection.P". The default setting is "garbage_control(off)".
4. `garbage_window/1:` sets the period of time the general garbage collection looks back. The default setting is "garbage_window(-1)".
5. `garbage_window_step/1:` sets the period of time when general garbage collection is activated. The default setting is "garbage_window_step(-1)".
6. `logging/1:` enables logging all the events to output. Can take the values: "on" or "off" (defined in "utils.P"). The default setting is "logging(off)".
7. `logging_to_file/1:` enables logging to an external file. Can take values: "on" or "off" (defined in "utils.P"). Use "write_log/2" to log an event to a file The default setting is "logging_to_file(off)).

8. `output_temporary_files/1`: enables logging compiled event files (after binarization and compilation). Can have the values: "on" or "off" (defined in "event_utils.P"). The default setting is "output_temporary_files(off)).

9. `out_of_order/1`: enables detection of events out out of order. Can have the values: "on" or "off" (defined in "compiler.P"). The default setting is "out_of_order(off)".

10. `prolog_backend/1`: sets the prolog backend. Can be: "swi", "xsb", "yap" and "sicstus" (defined in "event_utils.P"). The default setting is "prolog_backend(swi)".

11. `revision_flag/1`: enables revision in detection of events. Can have the values: "on" or "off" (defined in "compiler.P"). The default setting is "revision_flag(off)".

12. `rule_sharing/1`: sets the sharing in execution between rule bodies. Can be: "on" and "off" (defined in "binrizer.P"). The default setting is "rule_sharing(off)".

13. `rule_sharing_debuging/1`: sets the sharing debuging. Can be: "on" and "off" (defined in "binrizer.P"). The default setting is "rule_sharing_debuging(off)".

14. `store_fired_events/1`: enables storing all the events fired. Can have the values: "on" or "off" (defined in "utils.P"). The default setting is "store_fired_events(off)".

15. `store_fired_events_java/1`: enables storing fired events for the Java interface Can have the values: "on" or "off" (defined in "java_interface.P"). The default setting is "store_fired_events_java(off)".

4.6 Foreign language interface

When ETALIS is used to build real-world systems, a foreign-language interface may be necessary to:

- combine ETALIS with existing programs and libraries, thereby forming composite systems;
- interface ETALIS with the widely used languages, like Java and C#.

ETALIS works with different Prolog systems: SWI, Yap, XSB and Sicstus Prolog. The user should contact these systems mailing lists for how to connect from external languages, such as, Java, C, C#. For instance, the mailing list SWI Prolog is here: `https://mailbox.iai.uni-bonn.de/mailman/listinfo.cgi/swi-prolog`. Please also use `news:comp.lang.prolog` (google group: `http://groups.google.com/group/comp.lang.prolog/topics`) for general Prolog questions.

General instructions on how to connect to ETALIS from an external language:

- In the external language (Java), start a SWI Prolog engine and keep it into a list/pool of running engines (sessionID,enginePointer).
- this engine should load ETALIS in Prolog: $['src/etalis.P']$. and set to store fired events: $set\_etalis\_flag(store\_fired$ to see what events were triggered in ETALIS. Although this flag has "java" in the name, it applies to all external languages.

- To load an event program from the external language into ETALIS, there are two options:
  - save the event program into a file and call $compile\_event\_file(File)$, or
  - send the rules to ETALIS using $compile\_event\_rules([RawEventRules])$.
- To ensure that the current program is the only one that the current ETALIS engine uses, first, you should call: $reset\_ETALIS.$ in the Prolog system, which resets ETALIS completely by deleting all event rules and all partial goals from memory.
- To fire a stream of events from the external language into ETALIS, the user should call $reset\_db$ in ETALIS to delete all the partial goals from memory.

  The stream of events should be put in a Prolog list, call: $fire\_event\_list\_return\_external\_events([EventList], Out$ and collect the results from OutputList. Repeat this operation for all your events and event streams.
- Finish ETALIS with "halt" in Prolog.

Specific instructions for connecting to ETALIS from Java. Each one of the Prolog systems that we used has different interfaces from Java:

- SWI has two: JPL and InterProlog
- Yap has one: InterProlog
- XSB has one: InterProlog
- Sicstus has one: PrologBeans

Notes for Java interfaces for SWI Prolog:

- Interprolog: `http://www.declarativa.com/interprolog`, `interprolog@declarativa.com`
- JPL (supports a single Prolog engine at a time in Java): `http://www.swi-prolog.org/packages/jpl`, `http://www.swi-prolog.org/Mailinglist.html`

The simplest way to connect to SWI from Java with Interprolog is to install SWI and add the path to the SWI executable (e.g., in Windows, "c:/Program Files/pl/bin/plcon.exe") in the environment varible $PATH$. In Linux, you can do this by adding the following to your $.bashrc$ file
$exportPATH = /path\_to\_SWIBIN : \$PATH$
and
$exportSWI\_BIN\_DIRECTORY = /path\_to\_SWIBIN$

For the Eclipse framework, add the path to the directory containing the executable SWI in the current classpath by using the Window menu -¿ Preferences -¿ Java -¿ BuildPath -¿ ClasspathVariables -¿ New (give it any name and the directories you want to add to classpath).

Please also put interprolog.jar in the CLASSPATH variable.

Chapter 5

The ETALIS system operands and standard predicates

5.1 ETALIS CEP operands

The syntax of ETALIS is taken from Prolog. The arguments of the ETALIS language are called *terms*. A *ETALIS term* can be constructed from any logical symbol or a term followed by any finite number of arguments. A *term* is either a *constant*, a *variable*, or a *compound term*.

A *constant* is either a *number* (integer or floating-point) or an *atom*. The printed form of an integer in ETALIS consists of a sequence of digits optionally preceded by a minus sign (`'-'`) interpreted as base 10 integers. A floating-point number consists of a sequence of digits with an embedded decimal point, optionally preceded by a minus sign (`'-'`). A atom is identified by its name, which is a sequence of up to 1000 characters (other than the null character). Variables may be written as any sequence of alphanumeric characters (including `'_'`) beginning with either a capital letter or `'_'`. Like in Prolog, the structured data objects of are *compound terms* (or *structures*). The external representation of a compound term comprises a *functor* (called the *principal functor* or the *name* of the compound term) and a sequence of one or more terms called *arguments*.

The operators in ETALIS are simply a notational convenience. However, from a practical or a programmer's point of view, the existence of operators is highly desirable.

Keeping in mind that, in Prolog, the operators of type `'xfy'` are *right-associative*, that is only the first (left-hand) subexpression must be of lower precedence; the right-hand subexpression can be of the same precedence as the main operator, and the *left-associative* operators (type `'yfx'`) are the other way around, the operators in ETALIS are:

```
op(1200,xfy, 'rule:'), % rule tags
op(1200,xfy, '<-'), % event rules
op(1025,yfx, 'seq'), % sequential conjunction
op(1025,yfx, 'forall_seq'), % forall sequential conjunction
 % (unrestricted consumption seq - can be combined with any
```

```
        % other comsumption policy)
       op(1040,yfx, 'and'), % classical conjunction
       op(1045,yfx, 'par'), % parallel conjunction
       op(1045,yfx, '#'),
       op(1053,yfx, 'or'), % disjunction
       op(1025,yfx, 'do'), % plugin for actions triggered by events
       op(1025,yfx, 'equals'), % Alan's interval operators
       op(1025,yfx, 'meets'),
       op(1025,yfx, 'during'),
       op(1025,yfx, 'starts'),
       op(1025,yfx, 'finishes'),
       op(1050,yfx, 'where'), % database conditional triggers
       op(1050,yfx, 'check'), % database conditional triggers with times: t1(Start
       op(1050,yfx, 'event_multiply'), % CEP event multiplication
       op(200,xf, 'star_times'), % implementation of *times
       op(1025,yfx, 'ntimes'), % implementation of N times
       op(1031,yfx, 'cnot'), % negation - interval absence
       op(1031,yfx, 'fnot'), % total negation - ``never happened''
... (other operators can be found in the ETALIS source file ``parser.P'')
```

5.2 ETALIS standard predicates

Whenever ETALIS is invoked, a large set of *standard* predicates are defined and can be called from the interpreter or other interfaces. Standard predicates are listed in this manual under the index heading *Standard predicates* and at an implementation level are declared in the directory `$ETALIS_DIR/src/`. The ETALIS system has several source files in the src directory.

1. `binarizer.P` contains the predicates to binarize the rules in the event programs. The $binarization(+EventRules,$ predicate takes a set of event rules and returns a set of binary rules, that is rules with at most two events and one operation in its body. Temporary events are created to generate the binary rules and the binarizer also enables sharing between rule bodies. Certain operands, like the negation or star_times, are binarized with specialized formulas. This file also contains predicates to log the binarized rules into temporary files: $logging\_binary\_file(+InputFile, +BinaryEventRules)$.

2. `compiler.P` contains the predicate $event2tr\_transformation(+BinaryEventRules, -TRRules)$ to transform event rules into transaction logic code. It also contains a set of special flags forspecifying the execution of event files:

   - $dynamic(out\_of\_order/1)$ enables or disables out-of-order complex event detection;
   - $dynamic(revision\_flag/1)$: enables or disables revision in complex event detection;
   - $dynamic(etalis\_justification/1)$: enables or disables justification.

3. `date_time.P` contains the predicates for date ad time manipulation. The $current\_datime(-D)$ returns the current date and time.

The $is\_datime(+T)$ predicate returns true if T is a datime $datime(Year, Month, Day, Hour, Min, Sec, ?Counter)$ with or without conter in date.
The $less\_datime(T1, T2)$ predicate compares datimes.
The $datime\_plus\_sec(+Datime1, +Seconds, -Datime2)$ adds a number of seconds to the $Datime1$.

The $datime\_minus\_sec(+Datime1, +Seconds, -Datime2)$ subtracts a number of seconds to the $Datime1$.
The $datime\_minus\_datime(+Datime, +Datime1, -Seconds)$ predicate computes the number of seconds between two datimes.

4. `etalis.P` contains the main predicates for event processing.
   The $compile\_event\_file(+File)$ predicate compiles an event file, parsing, binarization, transformation to transaction logic .
   The $event(+E)$ predicate executes one event triggering complex events where this event contributes .
   The $fire\_event\_list\_return\_external\_events(+EventList, -OutputList)$ predicate fires a list of events returning the flagged composed events .
   The $execute\_event\_stream\_file(+EventFile)$ predicate executes a stream of events from a file .
   The $ins\_event\_rule/1$ predicate inserts a new event rule .
   The $del\_event\_rule/1$ predicate deletes event rules where the attribute can be a rule reference or a rule. The reasons for using only the id in delete are the following:

   - event rules are compiled in multiple internal rules, so deleting all rules that resulted from one event rule is quite complicated: we have to compile the rule and extract all rules resulted from this one rule;
   - temporary events are re-named (and also new variables are generated) and these temporary have to be checked at deletion. Simple unification of even variant won't do it because right bindings have to be checked.
   - sharing is enabled and we don't want to delete common bodies that take part in other rules as well as the current one that we delete.
   - when we have rule ids (and properties associated to these ids) these are pushed into all compiled rules. We know exactly what to delete.
   - we can also do bulk deletes with ids: multiple rules can share the same ID (the id doesn't have to be unique), so deleting one ID deletes all rules that share that ID.

5. `event_utils.P` contains the predicates for logging, event consumption policy and spying events for debugging.
   The $logging\_TR\_file(+File, +TRRules)$ predicate writes the compiled event file to a file.
   The $spy\_event(EventFunctor/Arity)$ predicate marks $EventFunctor/Arity$ for debuging.
   The $print\_trigger(+EventSymbol/+Arity)$ predicate is a special predicate that the user can put in any event file. This predicate will mark the event for logging to the console everytime it is detected. This predicate can be used with uninstantiated $Event$, when it will print all the instances of any event symbol with the same number of arguments as stated in $print\_trigger/1$. Another special use of this

predicate is $print\_trigger(all\_defined\_events)$ which prints all events that are not temporary (that is, created by the binarization step).

6. `executor.P` contains the predicates for executing events.
   The $event\_trigger(+Event)$ predicate executes all the rules for a given event .

7. `flags.P` contains the predicates for chainging the ETALIS flags.
   The $set\_etalis\_flag(+Flag, +Value)$ predicate sets one of the etalis flags . See the Section 4.5 for more details.
   The $get\_etalis\_flag(+Flag, +Value)$ predicate gets one of the etalis flags .

8. `garbage_collection.P` contains the predicates for goal management.
   The $start\_garbage\_collection$ predicate starts the garbage collection .
   The $general\_garbage\_collection$ predicate starts the general garbage collection .
   The $pattern\_garbage\_collection$ predicate starts the pattern based garbage collection for $garbage\_control(pattern)$ where individual rules have GC time windows.

9. `java_interface.P` contains the predicates for interfacing with foreign languages. The $set\_etalis\_flag(store\_fir$ sets storing the return flagged composed events.

10. `justify_etalis.P` contains the predicates for justification and writing justification trees on the desktop and udraw.
    The $justify\_event(+Event, [T1, T2], -Justification)$, $justify\_event\_positive$ and $justify\_event\_negative$ predicates compute the justification for events.
    The $write\_justification(+J)$ predicate writes justification keeping some meaningful indentation.
    The $write\_justification\_udraw(File, J)$ predicate writes justification keeping some meaningful indentation

11. `labeled_event_rules.P` contains the predicates for checking conditions for labeled rules.
    The $check\_event\_rule\_conditions\_internal(+Label, +Head, [+T1, +T2], +ListERProperties)$ predicate checks all properties for the event rule with the given label and Head.

12. `logging.P` contains the predicates for logging.
    The $log(L)$ predicate logs an event.

13. `network_tcp_interface.P` contains the predicates for interfacing with the network
    The $create\_server(Port)$ predicate creates a TCP port.
    The $dispatch(AcceptFd)$ predicate dispatches a message.
    The $create\_client(Host, Port)$ predicate creates a TCP client on a port of a host server.

14. `parser.P` contains the predicates for parsing event rules.
    The $repeat\_read(+InputHandle, -RawEventRules)$ predicate reads from a file all the event rules.
    The $parse\_event\_rules/2$ predicate parses event rules into internal format.

15. `storage.P` contains the predicates for storing partial goals into memory.
    The $seeDB$ predicate prints the internal state of the database.
    The $reset\_db$ predicate deletes all the partial goals in the current state.

The $etr\_dbf(+Label, +Data)$, $etr\_insf(+Label, +Data)$ and $etr\_delf(+Label, +Data)$ query, insert and delete goals into the database.

16. `utils.P` contains the predicates for manipulation of sets, lists and other data structures usedin ETALIS.
    The $counter(+CounterName, +Value)$, $incCounter(+CounterName)$ and $resetCounter(+CounterName)$ predicates handle counters implemented in Prolog.
    The $set\_intersection/3$ and $set\_difference/3$ predicates handle sets implemented in Prolog. We assume that the elements are not duplicated inside the sets.

Chapter 6

Examples

In this chapter, we enumerate the ETALIS examples from the $examples$ directory in the distribution.

6.1 flower_delivery

```
% Implementation of the Flower Delivery Application (use case specification by
%       Dr. Opher Etzion, implementation by the Etalis team)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Phase 1: Bid Phase
%
% External basic events:
%       Delivery Request event is placed by a store in the system:
%                delivery_request/3
%                delivery_record(+StoreId,+ToCoordinates,+DeliveryTime)
%               where ToCoordinates is of the form:
%                coordinates(+SNHemisphere,+Latitude,+EWHemisphere,+Longitude)
%       GPS location (each van is equiped with a GPS modem which
%               periodically transmits a GPS location event):
%                gps_location/2
%                gps_location(+DriverId,+CurrentCoordinates)
%               where CurrentCoordinates is of the form:
%                coordinates(+SNHemisphere,+Latitude,+EWHemisphere,+Longitude)
%
% Database facts (defined below):
%        Store record:
%                store_record/3
%                store_record(+StoreId,+MinRankAccepted,+AssignmentPreference)
%               where MinRankAccepted is the minimum ranking of the driver that
%                the store is prepared to accept because each store has a
%                different level of tolerance for service quality,
%                AssignmentPreference is either "manual" or "automatic"
%        Driver records:
%                driver_record/2
%                driver_record(+DriverId,+Ranking)
%       Method to transform raw latitude and longitude values into the region
%               of the city the driver is currently in:
%                gps_to_region/2
%                gps_to_region(+Coordinates,-Region)
```

```
% Complex event: Delivery Request enriched with the minimum ranking that the
%       store is prepared to accept and with an DeliveryRequestId
% Note: Location is of the form: coordinates(+SNHemisphere,+Latitude,
%       +EWHemisphere,+Longitude)
% delivery_request_enriched/5
delivery_request_enriched(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,
                MinRank)<-
        delivery_request(StoreId,ToCoordinates,DeliveryTime) where
                (store_record(StoreId,MinRank,_AssignmentPreference),
                incCounter(delivery_request_counter),
                counter(delivery_request_counter,DeliveryRequestId)).
print_trigger(delivery_request_enriched/5).

% Multiplier: multiply the event "delivery_request_enriched" for each driver
% delivery_request_enriched_multiplied/6
delivery_request_enriched_multiplied(DeliveryRequestId,DriverId,StoreId,
                ToCoordinates,DeliveryTime,MinRank)<-
        delivery_request_enriched(DeliveryRequestId,StoreId,ToCoordinates,
                DeliveryTime,MinRank) event_multiply
        driver_record(DriverId,_Ranking).
        % event_multiply does not consume "delivery_request_enriched"
print_trigger(delivery_request_enriched_multiplied/6).

% Complex event: complex events which indicate in which region of the city the
%       driver is currently in, translated from raw latitude and longitude
% gps_location_translated/3
gps_location_translated(DriverId,Rank,Region)<-
        gps_location(DriverId,coordinates(SNHemisphere,Latitude,
                EWHemisphere,Longitude)) where
                ( driver_record(DriverId,Rank),
                gps_to_region(coordinates(SNHemisphere,Latitude,
                        EWHemisphere,Longitude),Region) ).
print_trigger(gps_location_translated/3).

% Complex event: Bid Request event is broadcasted to all drivers that pass the
%       filter for ranking and location.
% bid_request/5
bid_request(DeliveryRequestId,DriverId,StoreId,ToCoordinates,DeliveryTime)<-
        ( delivery_request_enriched_multiplied(DeliveryRequestId,DriverId,
                        StoreId,ToCoordinates,DeliveryTime,MinRank) and
                gps_location_translated(DriverId,Rank,Region) )
        where (=<(MinRank,Rank), gps_to_region(ToCoordinates,Region)).
print_trigger(bid_request/5).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Phase 2: Assignment Phase

% External basic events:
%  A driver responds to the Bid Request by sending a Delivery Bid event
%       designating his or her current location and comming pick up time.
%   delivery_bid/4
%   delivery_bid(+DeliveryRequestId,+DriverId,+CurrentCoordinates,
%       +PossiblePickupTime)
print_trigger(delivery_bid/4).

%  Two minutes after the broadcast the system starts the assignment process.
% Note: the waiting time is set by a configurable parameter in the database
%       (start_assignment_time/1). For instance, for streams we set it to 2sec.
%       because the stream is synthetic and we dont have to wait 2min.
% startAssignment/4
exceptionAlarm(startAssignment(DeliveryRequestId,StoreId,ToCoordinates,
                DeliveryTime),Time)<-
```

```
97          delivery_request_enriched(DeliveryRequestId,StoreId,ToCoordinates,
                    DeliveryTime,_MinRank) where (start_assignment_time(Time)).
99          % exceptionAlarm does not consume "delivery_request_enriched"
    print_trigger(startAssignment/4).
101
    % The assignment is either an automatic or a manual process, depending on the
103 %       stores preference.
    % start_automaticAssignment/4
105 start_automaticAssignment(DeliveryRequestId,StoreId,ToCoordinates,
                    DeliveryTime)<-
107         startAssignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime)
            where store_record(StoreId,_MinRank,automatic).
109 print_trigger(start_automaticAssignment/4).

111 % start_manualAssignment/4
    start_manualAssignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime)<-
113         startAssignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime)
            where store_record(StoreId,_MinRank,manual).
115 print_trigger(start_manualAssignment/4).

117 % If the process is automatic then the first bidder among the selected drivers
    %       wins the bid.
119 % The pickup time and delivery time are set and the Assignment event is sent to
    %       the driver.
121 % assignment/6
    % assignment(+DeliveryRequestId,+StoreId,+ToCoordinates,+DeliveryTime,+DriverId,
123 %       +ScheduledPickupTime)

125 consumable_pick_first(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,
                    MinRank)<-
127         delivery_request_enriched(DeliveryRequestId,StoreId,ToCoordinates,
            DeliveryTime,MinRank) where store_record(StoreId,_MinRank,automatic).
129 print_trigger(consumable_pick_first/5).

131 assignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,DriverId,
                    ScheduledPickupTime)<-
133         ((consumable_pick_first(DeliveryRequestId,StoreId,ToCoordinates,
                    DeliveryTime,MinRank) seq
135         delivery_bid(DeliveryRequestId,DriverId,CurrentCoordinates,
                    ScheduledPickupTime) ) and
137         start_automaticAssignment(DeliveryRequestId,StoreId,ToCoordinates,
                    DeliveryTime)).
139         % ) fnot no_bid_alert(DeliveryRequestId) % this line can be added in the
            %       code to specify that no bids are accepted after timeout (its
141         %       not addresed in the specification)
            % we do phase 5 separatelly as different events to show the different
143         %       phase, but it can also be done in this step
            %where ( ScheduledPickupTime=datime(Y,M,D,_,_,_),
145         %       incCounter(assignments(DriverId,date(Y,M,D))),
            %       counter(assignments(DriverId,date(Y,M,D)),Count),
147         %       write(Count),nl).
    print_trigger(assignment/6).
149
    % If the process is manual, the system collects the Delievery Bid events that
151 %       match the original Bid Request and sends the five highest-ranked of
    %       these to the store.
153 % manualAssignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime)
    % collect_highest_five_delivery_bids/5
155 collect_highest_five_delivery_bids(DeliveryRequestId,StoreId,ToCoordinates,
                    DeliveryTime,[])<- % initialize with empty list
157         delivery_request_enriched(DeliveryRequestId,StoreId,ToCoordinates,
                    DeliveryTime,_MinRank) where
159         store_record(StoreId,_MinRank,manual).
```

```
collect_highest_five_delivery_bids(DeliveryRequestId,StoreId,ToCoordinates,
                DeliveryTime,HighestFive)<-
        ( collect_highest_five_delivery_bids(DeliveryRequestId,StoreId,
                ToCoordinates,DeliveryTime,TempHighestFive) seq
        delivery_bid(DeliveryRequestId,DriverId,CurrentCoordinates,
                PossiblePickupTime) ) where
        (driver_record(DriverId,Rank),
        select_highest_five([driver(DriverId,Rank,PossiblePickupTime)|
                TempHighestFive],HighestFive)).
print_trigger(collect_highest_five_delivery_bids/5).

% store_transmit_highest_five_delivery_bids/5 - event sent to the store
store_transmit_highest_five_delivery_bids(DeliveryRequestId,StoreId,
                ToCoordinates,DeliveryTime,HighestFive)<-
        ( collect_highest_five_delivery_bids(DeliveryRequestId,StoreId,
                ToCoordinates,DeliveryTime,HighestFive) and
        start_manualAssignment(DeliveryRequestId,StoreId,ToCoordinates,
                DeliveryTime) )
        where (HighestFive\=[]).
print_trigger(store_transmit_highest_five_delivery_bids/5).

assignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,DriverId,
                ScheduledPickupTime)<-
        store_transmit_highest_five_delivery_bids(DeliveryRequestId,StoreId,
                ToCoordinates,DeliveryTime,HighestFive) seq
        store_select_delivery_bid(DeliveryRequestId,DriverId,
                ScheduledPickupTime).
        % code to specify that no bids are accepted after timeout:
        % ) fnot no_bid_alert(DeliveryRequestId) )
        % ) fnot no_choice_alert(DeliveryRequestId).

% Alerts:
% If there are no bidders an alert is sent both to the store and the system
%       manager;
% no_bid_alert/1
no_bid_alert(DeliveryRequestId)<-
        start_automaticAssignment(DeliveryRequestId,StoreId,ToCoordinates,
                DeliveryTime) fnot
        delivery_bid(DeliveryRequestId,_DriverId,_CurrentCoordinates,
                _PossiblePickupTime).
no_bid_alert(DeliveryRequestId)<-
        start_manualAssignment(DeliveryRequestId,StoreId,ToCoordinates,
                DeliveryTime) fnot
        delivery_bid(DeliveryRequestId,_DriverId,_CurrentCoordinates,
                        _PossiblePickupTime).
print_trigger(no_bid_alert/1).

% If the store has not performed its manual assigment within one minute of
%       receiving its Delivery Bid events then both the store and the system
%       manager receive and alert.
% check_manual_assignment/4
exceptionAlarm(check_manual_assignment(DeliveryRequestId,StoreId,ToCoordinates,
                DeliveryTime),Time)<-
        store_transmit_highest_five_delivery_bids(DeliveryRequestId,StoreId,
                ToCoordinates,DeliveryTime,HighestFive) where
        (check_manual_assignment_time(Time)).
print_trigger(check_manual_assignment/4).

% no_choice_alert/1
no_choice_alert(DeliveryRequestId)<-
        check_manual_assignment(DeliveryRequestId,StoreId,ToCoordinates,
                DeliveryTime) fnot
        store_select_delivery_bid(DeliveryRequestId,_DriverId,
```

```
                _PossiblePickupTime).
print_trigger(no_choice_alert/1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Phase 3: Delivery Process

% External basic events:
%When the driver arrives to pick up the flowers the store sends a
%       pick_up_confirmation event:
% pick_up_confirmation/3
% pick_up_confirmation(+DeliveryRequestId,+DriverId,+RealPickupTime)
print_trigger(pick_up_confirmation/3).
%When the driver delivers the flowers the person receiving them confirms by
%       signing the drivers mobile device, and this generates a
%       delivery_confirmation event:
% delivery_confirmation/3).
% delivery_confirmation(+DeliveryRequestId,+DriverId,+RealDeliveryTime)
print_trigger(delivery_confirmation/3).

%Both pick_up_confirmation and delivery_confirmation events have time-stamps
%       associated with them, and this allows the system to generate several
%       alert events.
%A pick_up_alert is reported if a pick_up_confirmation has not been reported
%       within 5 minutes of the committed pick up time.
% check_pick_up/4
exceptionAlarmAbsoluteDatime(check_pick_up(DeliveryRequestId,StoreId,DriverId,
                ScheduledPickupTime),CheckTimePickup)<-
        assignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,
                DriverId,ScheduledPickupTime) where
        % the wait time is a configurable paramenter and can be changed in
        %       the test database
        (check_pick_up_time(WaitDuration),
        addSec_Datime(WaitDuration,ScheduledPickupTime,CheckTimePickup)).
print_trigger(check_pick_up/4).

% pick_up_alert/4
pick_up_alert(DeliveryRequestId,StoreId,DriverId,ScheduledPickupTime)<-
        ( check_pick_up(DeliveryRequestId,StoreId,DriverId,
                ScheduledPickupTime) fnot
        % the delivery was not handed over to another driver
        handover(DeliveryRequestId,DriverId,_DriverIdB) ) fnot
        pick_up_confirmation(DeliveryRequestId,DriverId,_RealPickupTime).
print_trigger(pick_up_alert/4).

%A delivery_alert is reported if a delivery_confirmation has not been reported
%       within ten minutes of the commited delivery time.
% check_delivery/4
exceptionAlarmAbsoluteDatime(check_delivery(DeliveryRequestId,StoreId,DriverId,
        DeliveryTime),CheckTimeDelivery)<-
        assignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,
                DriverId,_ScheduledPickupTime) where
        (check_delivery_time(WaitDuration),
                addSec_Datime(WaitDuration,DeliveryTime,CheckTimeDelivery)).
print_trigger(check_delivery/4).
% delivery_alert/4
delivery_alert(DeliveryRequestId,StoreId,DriverId,DeliveryTime)<-
        ( ( check_delivery(DeliveryRequestId,StoreId,DriverId,
                DeliveryTime) fnot
        % the delivery was not handed over to another driver
        handover(DeliveryRequestId,DriverId,_DriverIdB) ) fnot
        % there was no one to receive the package
        no_one_to_receive(DeliveryRequestId)
        ) fnot
```

```
        delivery_confirmation(DeliveryRequestId,DriverId,_RealDeliveryTime).
print_trigger(delivery_alert/4).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Phase 4: Ranking evaluation VERSION 1 (uses internal database for CEP)
%The system performas an evaluation of each driver ranking every time that
%       driver completes 20 deliveries.
%If the driver did not have any Delivery Alerts during that period then the
%       system generates a Ranking Increase event indicating that the drivers
%       ranking has increased by one point.
%Conversely if the driver has had more than five delivery alerts during that
%       time then the system generates a Ranking Decrease to reduce the ranking
%       by one point.

% counting_driver_deliveries/2
counting_driver_deliveries(DriverId,NewCount)<-
        delivery_confirmation(DeliveryRequestId,DriverId,_DeliveryTime) where
        (ranking_threshold(Max),
                counter(driver(DriverId),Count),
                Count<Max,
                incCounter(driver(DriverId)),
                counter(driver(DriverId),NewCount)).
print_trigger(counting_driver_deliveries/2).

%faulted_ranking/2
faulted_ranking(DriverId)<-
        delivery_alert(_DeliveryRequestId,_StoreId,DriverId,_DeliveryTime)
        where ( incCounter(faulted(DriverId)) ).
print_trigger(faulted_ranking/1).

% ranking_decrease/2
ranking_decrease(DriverId,NewRank)<-
        delivery_confirmation(DeliveryRequestId,DriverId,_DeliveryTime)
        where (ranking_threshold(Max), counter(driver(DriverId),Count),
               Count=Max, counter(faulted(DriverId),CountAlarms),
               CountAlarms>=5, resetCounter(driver(DriverId)),
               driver_record(DriverId,Rank), NewRank is Rank-1,
               retract(driver_record(DriverId,Rank)),
               assert(driver_record(DriverId,NewRank)),
               resetCounter(faulted(DriverId)),
               set_flag(precedent_decrease(DriverId),yes) ).
print_trigger(ranking_decrease/2).

% ranking_increase/2
ranking_increase(DriverId,NewRank)<-
        delivery_confirmation(DeliveryRequestId,DriverId,_DeliveryTime)
        where ( ranking_threshold(Max), counter(driver(DriverId),Count),
               Count=Max, counter(faulted(DriverId),CountAlarms),
               CountAlarms=0, resetCounter(driver(DriverId)),
               driver_record(DriverId,Rank), NewRank is Rank+1,
               retract(driver_record(DriverId,Rank)),
               assert(driver_record(DriverId,NewRank)) ).
print_trigger(ranking_increase/2).

% If the generation for a Ranking Increase was for a driver, whose previous
%       evaluation generated a Ranking Decrease in the previous evaluation, then
%       the system generates an Improvement Note.
%improvement_note/1
improvement_note(DriverId)<-
        ranking_increase(DriverId,_NewRank)
        where (get_flag(precedent_decrease(DriverId),yes),
                set_flag(precedent_decrease(DriverId),nil)).
print_trigger(improvement_note/1).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Phase 4: Ranking evaluation VERSION 2 (uses event operators to compose CEP)
%%The system performas an evaluation of each driver ranking every time that
%%        driver completes 20 deliveries.
%%If the driver did not have any Delivery Alerts during that period then the
%%        system generates a Ranking Increase event indicating that the drivers
%%        ranking has increased by one point.
%%Conversely if the driver has had more than five delivery alerts during that
%%        time then the system generates a Ranking Decrease to reduce the ranking
%%        by one point.
%
%% a single event is enough to start the counting for deliveries for all drivers
%%        registered in store1
%start_ranking_evaluation(DriverId, store1)<-
%         start_ranking_evaluation_for_all_drivers event_multiply
%         driver_record(DriverId, _Ranking).
%
%% driverEvaluationCounter/2
%driverEvaluationCounter(0, DriverId, StoreId)<-
%         start_ranking_evaluation(DriverId, StoreId).
%driverEvaluationCounter(Count, DriverId, StoreId)<- (
%         driverEvaluationCounter(CountTemp, DriverId, StoreId) seq
%         delivery_confirmation(DeliveryRequestId, DriverId, _DeliveryTime) )
%         where (Count is CountTemp+1).
%print_trigger(driverEvaluationCounter/3).
%
%% detects a driverEvaluation event every after every Max delivery.
%driverEvaluation(DriverId, StoreId)<-
%         driverEvaluationCounter(Count, DriverId, StoreId)
%         %Note: startRankingEvaluation will trigger driverEvaluation at the
%         %        beggining too.
%         where ( ranking_threshold(Max), (Count mod Max)=:=0 ).
%print_trigger(driverEvaluation/2).
%
%ranking_increase(DriverId, NewRank)<-
%         ( ( driverEvaluation(DriverId, StoreId)
%                 % enumerates for correct threashold
%                 %seq driverEvaluation(DriverId, StoreId)
%                 ) cnot
%         delivery_alert(DeliveryRequestId, StoreId, DriverId, DeliveryTime) ) where
%         ( driver_record(DriverId, Rank), NewRank is Rank+1,
%         retract(driver_record(DriverId, Rank)),
%         assert(driver_record(DriverId, NewRank)) ).
%print_trigger(ranking_increase/2).
%
%% counting_delivery_alerts/2
%counting_delivery_alerts(0, DriverId, StoreId)<-
%         driverEvaluation(DriverId, StoreId).
%counting_delivery_alerts(Count, DriverId, StoreId)<-
%         ( counting_delivery_alerts(CountTemp, DriverId, StoreId) seq
%         delivery_alert(DeliveryRequestId, StoreId, DriverId, DeliveryTime) )
%         where (Count is CountTemp+1).
%print_trigger(counting_delivery_alerts/3).
%
%% Detects when number of delivery_alert events exceeds defined maximum
%% (i.e. defined in delivery_alarm_threshold(Max)):
%ranking_decrease(DriverId, NewRank)<-
%         (( counting_delivery_alerts(Count, DriverId, StoreId) where (
%                 delivery_alarm_threshold(Max), Count >= Max ) )
%         seq driverEvaluation(DriverId, StoreId)) where
%         ( driver_record(DriverId, Rank), NewRank is Rank-1,
%                 retract(driver_record(DriverId, Rank)),
```

```
%                    assert(driver_record(DriverId,NewRank))).
%print_trigger(ranking_decrease/2).
%
%% improvement_note can be, in general, detected with the two following rules:
%neutral_note(DriverId)<-
%          ( driverEvaluation(DriverId, StoreId) seq
%          driverEvaluation(DriverId, StoreId) ) cnot
%          ( ranking_increase(DriverId, Rank) or ranking_decrease(DriverId, Rank) ).
%print_trigger(neutral_note/1).
%
%improvement_note(DriverId)<-
%          ( ranking_decrease(DriverId, Rank1) seq
%                    ranking_increase(DriverId, Rank2) ) cnot
%          neutral_note(DriverId).
%print_trigger(improvement_note/1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%Phase 5: Activity Monitoring VERSION 1 (uses the internal database for
%%          statistics)
%%The system generates aggregates assignment and other events and counts the
%%          number of assignments per day for each driver for each day on which the
%%          driver has been active.
%% Once a month the system creates reports on drivers performance, asserting the
%%          drivers according to the following criteria:
%%- A permanent weak driver is a driver with fewer than five assignments on all
%%          the days on which the driver was active.
%%- An idle driver is a driver with at least one day of activity which has no
%%          assignments.
%%- A consistent weak driver is a driver, whose daily assignments are at least
%%          two standard deviations lower than the average assignment per driver on
%%          each day in question.
%%- A consistent strong driver is a driver, whose daily assignments are at least
%%          two standard deviations higher than the average assignment per driver on
%%          each day in question.
%%- An improving driver is a driver whose assignments increase or stay the same
%%          day by day.
%
%% All the above are queries, not events, so they are treated in the
%%          "flower_specification_static_rules.P" file. They can also be specified
%%          here with "db/1" facts.
%
%keep_counter<-
%          % Note: the event can be done in the same step with assignment
%          assignment(DeliveryRequestId, StoreId, ToCoordinates, DeliveryTime,
%                    DriverId, ScheduledPickupTime)
%          where ( ScheduledPickupTime=datime(Y,M,D,_,_,_),
%                    incCounter(assignments(DriverId, date(Y,M,D))),
%                    retractall(work_day(DriverId, date(Y,M,D))),
%                    assert(work_day(DriverId, date(Y,M,D))) ).
%
%keep_counter_bids<-
%          bid_request(DeliveryRequestId, DriverId, StoreId, ToCoordinates,
%                    DeliveryTime)
%          where ( DeliveryTime=datime(Y,M,D,_,_,_),
%                    incCounter(bids(DriverId, date(Y,M,D))),
%                    retractall(work_day(DriverId, date(Y,M,D))),
%                    assert(work_day(DriverId, date(Y,M,D))) ).
%
%% report_event/1
%report_event(report(month(Y,M),L1,L2,L3,L4,L5))<-
%          end_month(month(Y,M)) where
%          (monthly_report(month(Y,M),L1,L2,L3,L4,L5)).
%print_trigger(report_event/1).
```

```
475
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
477 %Phase 5: Activity Monitoring VERSION 2 (uses event operators for statistics)
    %
479 % We implement the Activity Monitoring phase using system events that are
    %       triggered periodically, i.e. month(date(Y,M)) is triggered at the
481 %       beginning of each month; and day(date(D)) is triggered at
    %       the end of each working day.
483 % Further on, the Activity Monitoring is implemented only for one driver.
    %       There should be similar rules created for each registered driver, and
485 %       multiplication used to multiply events.
    % Multiplier: multiply the event "driver_activity_monitoring" for each driver
487 %        driver_activity_monitoring(DriverId,_Ranking)<-
    %                day event_multiply driver_record(DriverId,_Ranking).
489 %                print_trigger(driver_activity_monitoring/2).

491 % A driver, whenever starting a new working day, is expected to send an
    %       activeDriver(DriverID,StoreId,date(Y,M,D))
493 % event. The event contains a driver ID, a store ID and a date stamp.
    % assignmentCounter/2 counts no. of assignments per each driver (and store)
495 % per day:
    assignmentCounter(0,DriverId,StoreId,date(Y,M,D))<-
497         activeDriver(DriverId,StoreId,date(Y,M,D)).
    assignmentCounter(Count,DriverId,StoreId,date(Y,M,D))<- (
499         assignmentCounter(CountTemp,DriverId,StoreId,date(Y,M,D)) seq
            assignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,
501                 DriverId,ScheduledPickupTime) )
            where (Count is CountTemp+1).
503 print_trigger(assignmentCounter/4).

505 highActivity(DriverId,date(Y,M,D))<-
            assignmentCounter(Count,DriverId,StoreId,date(Y,M,D)) seq day(date(D))
507         where Count>=5.
    print_trigger(highActivity/2).
509
    permanentWeakDriver(DriverId,date(Y,M))<-
511         ( month(date(Y,M)) seq month(date(Y,M1))) cnot
            highActivity(DriverId,date(_,_,_)).
513 print_trigger(permanentWeakDriver/2).

515 zeroActivity(DriverId,date(D))<-
            ( day(date(D)) seq day(date(D1)) ) cnot
517         assignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,
                    DriverId,ScheduledPickupTime).
519 print_trigger(zeroActivity/2).

521 idleDriver(DriverId,date(Y,M))<-
            ( month(date(Y,M)) seq zeroActivity(DriverId,date(_)) seq
523         month(date(Y,M1)) ).
    print_trigger(idleDriver/2).
525
    strongActivity(DriverId,date(Y,M,D))<-
527         assignmentCounter(Count,DriverId,StoreId,date(Y,M,D)) seq day(date(D))
            % instead of Count > Avg, we should calculate 2 standard deviations
529         %       lower than Avg
            where (average_driver_assignment(Avg), Count > Avg).
531
    print_trigger(strongActivity/2).
533
    consistentWeakDriver(DriverId,date(Y,M))<-
535         (month(date(Y,M)) seq month(date(Y,M1))) cnot
            strongActivity(DriverId,date(_,_,_)).
537 print_trigger(consistentWeakDriver/2).
```

```
weakActivity(DriverId,date(Y,M,D))<-
        assignmentCounter(Count,DriverId,StoreId,date(Y,M,D)) seq day(date(D))
        % instead of Count > Avg, we should calculate 2 standard deviations
        %       higher than Avg
        where (average_driver_assignment(Avg), Count < Avg).

print_trigger(weakActivity/2).

consistentStrongDriver(DriverId,date(Y,M))<-
        (month(date(Y,M)) seq month(date(Y,M1))) cnot
        weakActivity(DriverId,date(_,_,_)).
print_trigger(consistentStrongDriver/2).

activity(DriverId,date(Y,M,D),Count)<-
        assignmentCounter(Count,DriverId,StoreId,date(Y,M,D)) seq day(date(D)).
print_trigger(activity/3).

decreasingActivity(DriverId,date(Y,M))<-
        activity(DriverId,date(Y,M,D1),Count1) seq
        activity(DriverId,date(Y,M,D2),Count2)
        where Count1 > Count2.
print_trigger(decreasingActivity/2).

improvingDriver(DriverId,date(Y,M))<-
        (month(date(Y,M)) seq month(date(Y,M1))) cnot
        decreasingActivity(DriverId,date(_,_)).
print_trigger(improvingDriver/2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Additional rules

%1. Handover - When driver A decides to pass the assignment to another
%driver (B) he produces "handover" event and now driver B is
%responsible for the delivery.
%    a. The event can be produced only if the first driver was chosen
%automatically (not manually) by the store.
%    b. The second driver that receives the assignment from the first
%will have the same (or higher) rank than the first one.

% basic event
% handover(+DeliveryRequestId,+DriverA,+DriverB)

% change assignment if handover/3 detected
assignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,
                DriverIdB,ScheduledPickupTime)<-
        assignment(DeliveryRequestId,StoreId,ToCoordinates,DeliveryTime,
                DriverIdA,ScheduledPickupTime) seq
        handover(DeliveryRequestId,DriverIdA,DriverIdB) where
        (
                store_record(StoreId,_MinRank,automatic),
                driver_record(DriverIdA,Ranking1),
                driver_record(DriverIdB,Ranking2),
                Ranking1 =< Ranking2
        ).

%2. No one to receive - When driver delivers flowers and there is no
%one to receive the flowers at recipient address, he decides to drop it
%at porch or to drop at neighbor (depends on the area). And uses his
%mobile device that sends No one to receive alert to the system, and
%this way Confirmation Delivery alert is canceled.

% basic event
```

```
% no_one_to_receive(+DeliveryRequestId)

% modifies delivery_alert(DeliveryRequestId, StoreId, DriverId, DeliveryTime) above
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Static Rules for the flower_specification use case
flower_use_case_interface:-
        flower_use_case_write_interface_instructions,
        repeat_read_instruction.

flower_use_case_write_interface_instructions:-
        nl, nl,
        write('    Flower use case instructions: introduce events with '),
        write('"event(EventInstance)." and exit with "halt."'), nl, nl.

repeat_read_instruction:-
        read_term(Term,[]),
        call(Term),
        repeat_read_instruction.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Phase 5: Activity Monitoring
%- A permanent weak driver is a driver with fewer than five assignments on all
%       the days on which the driver was active.
permanent_weak_driver(month(Y,M),DriverId):-
        driver_record(DriverId,_Rank),
        findall( day_report(DriverId,date(Y,M,D),Count),
               ( work_day(DriverId,date(Y,M,D)),
               counter(assignments(DriverId,date(Y,M,D)),Count) ), L),
        permanent_weak_driver_internal(L).
permanent_weak_driver_internal([day_report(_DriverId,date(_Y,_M,_D),Count)]):-
        Count\=0,Count<5.
permanent_weak_driver_internal([day_report(_DriverId,date(_Y,_M,_D),Count)|T]):-
        Count\=0,Count<5,
        permanent_weak_driver_internal(T).

%- An idle driver is a driver with at least one day of activity which has no
%       assignments.
idle_driver(month(Y,M),DriverId):-
        driver_record(DriverId,_Rank),
        findall( day_report(DriverId,date(Y,M,D),BidCount,Count),
                ( work_day(DriverId,date(Y,M,D)),
                counter(bids(DriverId,date(Y,M,D)),BidCount),
                counter(assignments(DriverId,date(Y,M,D)),Count) ), L),
        my_member(X,L),
        idle_driver_internal(X).
idle_driver_internal(day_report(_DriverId,date(_Y,_M,_D),BidCount,Count)):-
        BidCount>0, Count=0.

%- A consistent weak driver is a driver, whose daily assignments are at least
%       two standard deviations lower than the average assignment per driver on
%       each day in question.
consistent_weak_driver(month(Y,M),DriverId):-
        driver_record(DriverId,_Rank),
        findall( day_report(DriverId,date(Y,M,D),Count),
                ( work_day(DriverId,date(Y,M,D)),
                counter(assignments(DriverId,date(Y,M,D)),Count) ), L),
        consistent_weak_driver_internal(L).

consistent_weak_driver_internal([day_report(_DriverId,date(Y,M,D),Count)]):-
        average_assignment(date(Y,M,D),Avg), Avg2 is Avg-2,
```

```
        Count=<Avg2.
consistent_weak_driver_internal([day_report(_DriverId,date(Y,M,D),Count)|T]):-
        average_assignment(date(Y,M,D),Avg), Avg2 is Avg-2,
        Count=<Avg2,
        consistent_weak_driver_internal(T).

average_assignment(date(Y,M,D),Avg):-
        % the date is given, so the findall finds all day_reports for one day
        %       from all drivers
        findall( day_report(DriverId,date(Y,M,D),Count),
                ( driver_record(DriverId,_Rank),
                work_day(DriverId,date(Y,M,D)),
                counter(assignments(DriverId,date(Y,M,D)),Count) ), L),
        sum_all(L,Sum),
        my_length(L,Size),
        (Size>0 -> Avg is Sum/Size ; Avg=0),
        !.

sum_all(L,Sum):-
        sum_all(L,0,Sum).
sum_all([],Sum,Sum).
sum_all([day_report(_DriverId,_Date,Count)|T],PartialSum,Sum):-
        NewPartialSum is PartialSum+Count,
        sum_all(T,NewPartialSum,Sum).

%- A consistent strong driver is a driver, whose daily assignments are at least
%       two standard deviations higher than the average assignment per driver on
%       each day in question.
consistent_strong_driver(month(Y,M),DriverId):-
        driver_record(DriverId,_Rank),
        findall( day_report(DriverId,date(Y,M,D),Count),
                ( work_day(DriverId,date(Y,M,D)),
                counter(assignments(DriverId,date(Y,M,D)),Count) ), L),
        consistent_strong_driver_internal(L).

consistent_strong_driver_internal([day_report(_DriverId,date(Y,M,D),Count)]):-
        average_assignment(date(Y,M,D),Avg), Avg2 is Avg+2,
        Count>=Avg2.
consistent_strong_driver_internal([day_report(_DriverId,date(Y,M,D),Count)|T]):-
        average_assignment(date(Y,M,D),Avg), Avg2 is Avg+2,
        Count>=Avg2,
        consistent_strong_driver_internal(T).

%- An improving driver is a driver whose assignments increase or stay the same
%       day by day.
improving_driver(month(Y,M),DriverId):-
        % the days are ordered in they were asserted
        driver_record(DriverId,_Rank),
        findall( day_report(DriverId,date(Y,M,D),Count),
                ( work_day(DriverId,date(Y,M,D)),
                counter(assignments(DriverId,date(Y,M,D)),Count) ), L),
        improving_driver_internal(L).

improving_driver_internal([_]).
improving_driver_internal([day_report(DriverId,_,Count1),
                day_report(DriverId,_,Count2)|T]):-
        Count1=<Count2,
        improving_driver_internal([day_report(DriverId,_,Count2)|T]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% ?- monthly_report(month(Y,M),L1,L2,L3,L4,L5),
%       write(report(month(Y,M),L1,L2,L3,L4,L5)),nl,nl.
monthly_report(month(Y,M),L1,L2,L3,L4,L5):-
```

```
        findall( permanent_weak_driver(DriverId),
                permanent_weak_driver(month(Y,M),DriverId), L1 ),
        findall( idle_driver(DriverId), idle_driver(month(Y,M),DriverId), L2 ),
        findall( consistent_weak_driver(DriverId),
                consistent_weak_driver(month(Y,M),DriverId), L3 ),
        findall( consistent_strong_driver(DriverId),
                consistent_strong_driver(month(Y,M),DriverId), L4 ),
        findall( improving_driver(DriverId),
                improving_driver(month(Y,M),DriverId), L5 ),
        !.
:- dynamic(work_day/2).

% select_highest_five/2
% select_highest_five(+L1,-L2) where L1 is of the form
%       [driver(DriverId,Rank,PickupTime),...]
select_highest_five(L1,L2) :- select_highest_five(L1,[],L2).
% select_highest_five/3
% select_highest_five(+L1,+TempResult,-L2)
select_highest_five([],L,L).
select_highest_five([driver(D,R,P)|T],[],L) :-
        select_highest_five(T,[driver(D,R,P)],L).
select_highest_five([driver(D,R,P)|T],[driver(D1,R1,P1)],L) :-
        select_highest_five(T,[driver(D,R,P),driver(D1,R1,P1)],L).
select_highest_five([driver(D,R,P)|T],[driver(D1,R1,P1),driver(D2,R2,P2)],L) :-
        select_highest_five(T,[driver(D,R,P),driver(D1,R1,P1),
                driver(D2,R2,P2)],L).
select_highest_five([driver(D,R,P)|T],[driver(D1,R1,P1),driver(D2,R2,P2),
                driver(D3,R3,P3)],L) :-
        select_highest_five(T,[driver(D,R,P),driver(D1,R1,P1),
                driver(D2,R2,P2),driver(D3,R3,P3)],L).
select_highest_five([driver(D,R,P)|T],[driver(D1,R1,P1),driver(D2,R2,P2),
                driver(D3,R3,P3),driver(D4,R4,P4)],L) :-
        select_highest_five(T,[driver(D,R,P),driver(D1,R1,P1),driver(D2,R2,P2),
                driver(D3,R3,P3),driver(D4,R4,P4)],L).
select_highest_five([driver(_D,R,_P)|T],[driver(D1,R1,P1),driver(D2,R2,P2),
                driver(D3,R3,P3),driver(D4,R4,P4),driver(D5,R5,P5)],L) :-
        R1>R,R2>R,R3>R,R4>R,R5>R,
        select_highest_five(T,[driver(D1,R1,P1),driver(D2,R2,P2),
                driver(D3,R3,P3),driver(D4,R4,P4),driver(D5,R5,P5)],L).
select_highest_five([driver(D,R,P)|T],[driver(_D1,R1,_P1),driver(D2,R2,P2),
                driver(D3,R3,P3),driver(D4,R4,P4),driver(D5,R5,P5)],L) :-
        R>R1,R2>R1,R3>R1,R4>R1,R5>R1,
        select_highest_five(T,[driver(D,R,P),driver(D2,R2,P2),driver(D3,R3,P3),
                driver(D4,R4,P4),driver(D5,R5,P5)],L).
select_highest_five([driver(D,R,P)|T],[driver(D1,R1,P1),driver(_D2,R2,_P2),
                driver(D3,R3,P3),driver(D4,R4,P4),driver(D5,R5,P5)],L) :-
        R1>R2,R>R2,R3>R2,R4>R2,R5>R2,
        select_highest_five(T,[driver(D1,R1,P1),driver(D,R,P),driver(D3,R3,P3),
                driver(D4,R4,P4),driver(D5,R5,P5)],L).
select_highest_five([driver(D,R,P)|T],[driver(D1,R1,P1),driver(D2,R2,P2),
                driver(_D3,R3,_P3),driver(D4,R4,P4),driver(D5,R5,P5)],L) :-
        R1>R3,R2>R3,R>R3,R4>R3,R5>R3,
        select_highest_five(T,[driver(D1,R1,P1),driver(D2,R2,P2),
                driver(D,R,P),driver(D4,R4,P4),driver(D5,R5,P5)],L).
select_highest_five([driver(D,R,P)|T],[driver(D1,R1,P1),driver(D2,R2,P2),
                driver(D3,R3,P3),driver(_D4,R4,_P4),driver(D5,R5,P5)],L) :-
        R1>R4,R2>R4,R3>R4,R>R4,R5>R4,
        select_highest_five(T,[driver(D1,R1,P1),driver(D2,R2,P2),
                driver(D3,R3,P3),driver(D,R,P),driver(D5,R5,P5)],L).
select_highest_five([driver(D,R,P)|T],[driver(D1,R1,P1),driver(D2,R2,P2),
                driver(D3,R3,P3),driver(D4,R4,P4),driver(_D5,R5,_P5)],L) :-
        R1>R5,R2>R5,R3>R5,R4>R5,R>R5,
        select_highest_five(T,[driver(D1,R1,P1),driver(D2,R2,P2),
```

```
                driver(D3,R3,P3),driver(D4,R4,P4),driver(D,R,P)],L).
```

```
swipl -g
    "['../../src/etalis.P'],set_etalis_flag(logging,off),
     compile_event_file('flower_specification.event'), load_static_rules('↩
         flower_specification_static_rules.P'), load_database('use_cases/flower_interface_01.db↩
         '), flower_use_case_interface."

################################################################################
rm -rf *.bin *.ctr
```

6.1.1 flower_delivery_01

```
% Flags and configuration parameters
start_assignment_time(2). % start assignment after 2 sec
check_manual_assignment_time(1). % check manual assignment after 1 sec
check_pick_up_time(3). % check pickup after 5 min
check_delivery_time(6). % check delivery after 10 min
ranking_threshold(4). % increment and decrement ranking for drivers after every 4 deliveries
delivery_alarm_threshold(20).

% Database for flower_test_01:
store_record('store1',5,automatic).
store_record('store2',6,manual).
store_record('store3',7,automatic).

driver_record('driver1',5).
driver_record('driver2',6).
driver_record('driver3',7).
driver_record('driver4',8).

gps_to_region(coordinates('N',X,'W',Y),'Manhattan')    :- 4042<X, X<4049, 7358<Y, Y<7370,!.
gps_to_region(coordinates('N',X,'W',Y),'TheBronx')     :- 4049<X,X<4059, 7352<Y,Y<7370,!.
gps_to_region(coordinates('N',X,'W',Y),'Brooklyn')     :- 4040<X,X<4042, 7358<Y,Y<7360,!.
gps_to_region(coordinates('N',X,'W',Y),'Queens')       :- 4042<X,X<4059, 7355<Y,Y<7364,!.
gps_to_region(coordinates('N',X,'W',Y),'StatenIsland') :- 4034<X,X<4040, 7368<Y,Y<7399,!.
```

```

%event(gps_location(driver1,coordinates('N',4043.100,'W',7359.100))).
%event(gps_location(driver2,coordinates('N',4043.200,'W',7359.200))).
%event(gps_location(driver3,coordinates('N',4043.300,'W',7359.300))).

%event(delivery_request(store1,coordinates('N',4044.000,'W',7360.000),datime↩
    (2009,12,24,10,30,0))).

%event(delivery_bid(1,driver1,coordinates('N',4043.100,'W',7359.100),datime(2009,12,24,10,10,0)↩
    )). % this is a synthetic stream (note: we know the delivery identifier generated by the ↩
    system for the delivery event)

%event(delivery_bid(1,driver2,coordinates('N',4043.200,'W',7359.200),datime(2009,12,24,10,20,0)↩
    )).

%sleep(3). % we leave enough time for the assignment to take place (2 sec)
```

```
%event(store_select_delivery_bid(1,driver2,datime(2009,12,24,10,2,0))).

%sleep(1).

%event(pick_up_confirmation(1,driver1,datime(2009, 12, 24, 10, 10, 0))).

%sleep(1).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
event(start_ranking_evaluation_for_all_drivers).

event(delivery_confirmation(1,driver1,datime(2009, 12, 24, 10, 20, 0))).
event(delivery_confirmation(2,driver1,datime(2009, 12, 24, 10, 20, 0))).
event(delivery_alert(3,store1,driver1,datime(2009, 12, 24, 10, 20, 0))).
event(delivery_confirmation(4,driver1,datime(2009, 12, 24, 10, 20, 0))).
event(delivery_confirmation(5,driver1,datime(2009, 12, 24, 10, 20, 0))).

event(delivery_confirmation(6,driver2,datime(2009, 12, 25, 10, 20, 0))).

event(delivery_confirmation(7,driver1,datime(2009, 12, 25, 10, 20, 0))).
event(delivery_confirmation(8,driver1,datime(2009, 12, 25, 10, 20, 0))).
event(delivery_confirmation(9,driver1,datime(2009, 12, 25, 10, 20, 0))).
event(delivery_confirmation(10,driver1,datime(2009, 12, 25, 10, 20, 0))).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%event(assignment(11,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver1,datime(2009,12,24,11,40,0))).
%event(assignment(12,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,12,30,0),driver1,datime(2009,12,24,12,40,0))).
%event(assignment(13,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,13,30,0),driver1,datime(2009,12,24,13,40,0))).
%event(assignment(14,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,14,30,0),driver1,datime(2009,12,24,14,40,0))).
%event(assignment(15,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,15,30,0),driver1,datime(2009,12,24,15,40,0))).

%event(assignment(16,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,25,11,30,0),driver1,datime(2009,12,25,11,40,0))).
%event(assignment(17,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,25,12,30,0),driver1,datime(2009,12,25,12,40,0))).
%event(assignment(18,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,25,13,30,0),driver1,datime(2009,12,25,13,40,0))). % SWI Prolog has a limited number of alarms under Windows. However, there can be created 100,000 alarms under Linux.
%event(assignment(19,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,25,14,30,0),driver1,datime(2009,12,25,14,40,0))).
%event(assignment(20,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,25,15,30,0),driver1,datime(2009,12,25,15,40,0))).

%permanent_weak_driver(driver1) -> ( write('permanent_weak_driver(driver1)'),nl ); true.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%event(bid_request(21,driver1,store1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0))).
%idle_driver(driver1) -> ( write('idle_driver(driver1)'),nl ) ; true.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%event(assignment(11,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver2,datime(2009,12,24,11,40,0))).
%event(assignment(12,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver2,datime(2009,12,24,11,40,0))).
%event(assignment(13,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver2,datime(2009,12,24,11,40,0))).
```

```
%event(assignment(14,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver2,datime(2009,12,24,11,40,0))).
%event(assignment(15,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver2,datime(2009,12,24,11,40,0))).
%event(assignment(16,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver2,datime(2009,12,24,11,40,0))).

%event(assignment(17,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver1,datime(2009,12,24,11,40,0))).

%consistent_weak_driver(driver1) -> ( write('consistent_weak_driver(driver1)'),nl ); true.
%consistent_strong_driver(driver1) -> ( write('consistent_weak_driver(driver1)'),nl ); true.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%event(assignment(31,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,23,11,30,0),driver1,datime(2009,12,23,11,40,0))).
%event(assignment(32,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,23,11,30,0),driver1,datime(2009,12,23,11,40,0))).
%event(assignment(32,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver1,datime(2009,12,24,11,40,0))).
%event(assignment(33,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver1,datime(2009,12,24,11,40,0))).
%improving_driver(driver1) -> ( write('improving_driver(driver1)'),nl ); true.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%44%% event(assignment(31,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,23,11,30,0),driver1,datime(2009,12,23,11,40,0))).
%%44%% event(assignment(32,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,23,11,30,0),driver1,datime(2009,12,23,11,40,0))).
%%44%% event(assignment(32,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver1,datime(2009,12,24,11,40,0))).
%%44%% event(assignment(33,1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,24,11,30,0),driver1,datime(2009,12,24,11,40,0))).
%%44%% %monthly_report(month(2009,12),L1,L2,L3,L4,L5), nl,nl, write(report(month(2009,12),L1,L2,L3,L4,L5)),nl,nl.
%%44%% event(end_month(month(2009,12))).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Phase 4
event(startRankingEvaluation(driver2,store1)).
event(delivery_confirmation(1,driver2,datime(2009, 12, 24, 10, 20, 0))).
event(delivery_alert(1,store1,driver2,datime(2009, 12, 24, 10, 20, 0))).
event(counting_delivery_alerts(2,driver2,store1)).
event(driverEvaluation(driver2,store1)).
event(ranking_decrease(driver2,Rank)).
event(ranking_increase(driver2,Rank)).
event(neutral_note(driver2)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Phase 5
event(month(date(2009,11))).
event(activeDriver(driver2,store1,date(2009,11,10))).
event(assignment(31,store1,coordinates('N',4044.000,'W',7360.000),datime(2009,12,23,11,30,0),driver2,datime(2009,12,23,11,40,0))).
event(day(date(23))).
```

6.1.2 flower_delivery_02

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scenario 1 (normal flow):
```

```
% In the following scenarios we assume that there are 3 stores (numbered 1,2,3).
% The map, hence, is divided into three regions.
%
% Pre-conditions:
% Drivers A,B,C are in region 1.
%
% 00:00 Store 1 receives an order.
% 00:01 System matches drivers A,B,C  by location and ranking.
% 00:02 Store receives drives A,B,C and filters out driver C.
% 00:03 Bid request is sent to drivers A,B
% 00:04 Drivers A,B respond with delivery bid and provide their current location
%       and pickup time.
% 00:05 Driver A is chosen in automatic process by the store.
% 00:05 Assignment is sent to driver A.
% 00:10 A arrives to the store to pick-up the delivery. The store provides
%       pick-up confirmation.
% 00:15 A delivers the flowers to the customer. Delivery confirmation is
%       provided.
% 00:16 Ranking of A is increased by one.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Flags and configuration parameters
start_assignment_time(5). % start assignment in sec
                          % (2 min. in the specification)
check_manual_assignment_time(60). % check manual assignment in sec
                          % (1 min. in the specification)
check_pick_up_time(300). % check pickup after in sec
                          % (5 min. in the specification)
check_delivery_time(600). % check delivery after in sec
                          % (10 min. in the specification)
ranking_threshold(1). % increment/decrement ranking for drivers after N deliv.
                          % (20 deliveries in the specification)
delivery_alarm_threshold(5).
                          % (5 delivery alerts in the specification)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Database for scenarion 01:
store_record('store1',5,automatic).
store_record('store2',5,manual).
store_record('store3',5,automatic).

driver_record('driverA',7).
driver_record('driverB',6).
driver_record('driverC',3).

gps_to_region(coordinates('N',X,'W',Y),'Region_01') :-
        0=<X,X<100,
        0=<Y,Y<100,
        !.
gps_to_region(coordinates('N',X,'W',Y),'Region_02') :-
        100=<X,X<200,
        0=<Y,Y<100,
        !.
gps_to_region(coordinates('N',X,'W',Y),'Region_03') :-
        200=<X,X<300,
        0=<Y,Y<100,
        !.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scenario 1:
```

```
%       see flower_test_03.db
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Drivers location: all three drivers are located in the first region
event(gps_location(driverA,coordinates('N',10.000,'W',10.000))).
event(gps_location(driverB,coordinates('N',20.000,'W',20.000))).
event(gps_location(driverC,coordinates('N',30.000,'W',30.000))).

% Delivery request creation
event(delivery_request(store1,coordinates('N',10.000,'W',10.000),
        datime(2010,12,24,0,0,0))).

% Drivers send their bids
event(delivery_bid(1,driverA,coordinates('N',10.000,'W',10.000),
        datime(2010,12,24,0,4,0))).
event(delivery_bid(1,driverB,coordinates('N',20.000,'W',20.000),
        datime(2010,12,24,0,4,0))).

% Assignment of driverA is automatic at second 5
%event(assignment(1,store1,coordinates('N',50.000,'W',50.000),
%       datime(2010,12,24,0,15,0),driverA,datime(2010,12,24,0,10,0))).

% We leave enough time for the assignment to take place
sleep(6).

% DriverA picks up the delivery
event(pick_up_confirmation(1,driverA,datime(2010, 12, 24, 0, 10, 0))).

sleep(1).

% DriverA delivered in time
event(delivery_confirmation(1,driverA,datime(2010, 12, 24, 0, 15, 0))).

% Note: The ranking of the driverA is increased after just one delivery


%halt.
```

6.1.3 flower_delivery_03

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scenario 2:
% In the following scenarios we assume that there are 3 stores (numbered 1,2,3).
% The map, hence, is divided into three regions.
%
% Pre-conditions:
% Drivers E,F,G,H,I,J,K are in region 3
% Driver F had 10 delivery alerts during previous 19 deliveries.
%
% 00:00 Store 3 receives an order (System receives "delivery request" event).
%
% 00:01 System matches drivers F,G,H,I,J,K  by location and ranking (Driver E
%       is filtered out due     to his low rating).
%
% 00:02 Bid request is sent to drivers F,G,H,I,J,K .
%
% 00:03 Drivers F,G,H,I,J,K  respond with "delivery bid", provide their current
%       location and commit a pick-up time.
```

```
%
% 00:03-00:05. Store doesn't perform manual assignment (Alert is sent both to
%       the store and the system manager).
%
% 00:06 Driver F is chosen by the store (manual assignment is finally performed
%       after a  3 minute delay). Assignment is sent to driver F. Pick-up time
%       and delivery time is set.
%
% 00:10 Driver F arrives to the store while exceeding pick-up time by 6 minutes
%       (Pick-Up Alert expected). Pick-up confirmation is set.
%
% 00:25 F delivers the flowers to the customer while exceeding delivery time by
%       10 minutes (Delivery Alert expected). Delivery confirmation is provided.
%
% 00:31 Rank of F is decreased by one.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Flags and configuration parameters
start_assignment_time(2). % start assignment in sec
                                % (2 min. in the specification)
check_manual_assignment_time(3). % check manual assignment in sec
                                % (1 min. in the specification)
check_pick_up_time(2). % check pickup after in sec
                                % (5 min. in the specification)
check_delivery_time(3). % check delivery after in sec
                                % (10 min. in the specification)
ranking_threshold(20). % increment/decrement ranking for drivers after N deliv.
                                % (20 deliveries in the specification)
delivery_alarm_threshold(5).
                                % (5 delivery alerts in the specification)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Database for scenarion 01:
store_record('store1',2,manual).

driver_record('driverE',1).
driver_record('driverF',3).
driver_record('driverG',3).
driver_record('driverH',3).
driver_record('driverI',3).
driver_record('driverJ',3).
driver_record('driverK',3).

gps_to_region(coordinates('N',X,'W',Y),'Region_01') :-
        0=<X,X<100,
        0=<Y,Y<100,
        !.
gps_to_region(coordinates('N',X,'W',Y),'Region_02') :-
        100=<X,X<200,
        0=<Y,Y<100,
        !.
gps_to_region(coordinates('N',X,'W',Y),'Region_03') :-
        200=<X,X<300,
        0=<Y,Y<100,
        !.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Scenario 2:
%       see flower_test_03.db
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Pre-conditions:
% Drivers E,F,G,H,I,J,K are in region 3

% Drivers location: Drivers E,F,G,H,I,J,K are in region 3
event(gps_location(driverE,coordinates('N',210.000,'W',10.000))).
event(gps_location(driverF,coordinates('N',210.000,'W',10.000))).
event(gps_location(driverG,coordinates('N',210.000,'W',10.000))).
event(gps_location(driverH,coordinates('N',210.000,'W',10.000))).
event(gps_location(driverI,coordinates('N',210.000,'W',10.000))).
event(gps_location(driverJ,coordinates('N',210.000,'W',10.000))).
event(gps_location(driverK,coordinates('N',210.000,'W',10.000))).

% Driver F had 10 delivery alerts during previous 19 deliveries.
event(delivery_alert(-19,store1,driverF,datime(2010,1,1,0,0,0))).
event(delivery_alert(-18,store1,driverF,datime(2010,1,1,0,0,0))).
event(delivery_alert(-17,store1,driverF,datime(2010,1,1,0,0,0))).
event(delivery_alert(-16,store1,driverF,datime(2010,1,1,0,0,0))).
event(delivery_alert(-15,store1,driverF,datime(2010,1,1,0,0,0))).
event(delivery_alert(-14,store1,driverF,datime(2010,1,1,0,0,0))).
event(delivery_alert(-13,store1,driverF,datime(2010,1,1,0,0,0))).
event(delivery_alert(-12,store1,driverF,datime(2010,1,1,0,0,0))).
event(delivery_alert(-11,store1,driverF,datime(2010,1,1,0,0,0))).
event(delivery_alert(-10,store1,driverF,datime(2010,1,1,0,0,0))).
% delivery_confirmation(DeliveryRequestId,DriverId,DeliveryTime)
event(delivery_confirmation(-19,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-18,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-17,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-16,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-15,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-14,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-13,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-12,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-11,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-10,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-9,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-8,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-7,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-6,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-5,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-4,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-3,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-2,driverF,datime(2010,1,1,0,0,0))).
event(delivery_confirmation(-1,driverF,datime(2010,1,1,0,0,0))).

% 00:00 Store 3 receives an order (System receives "delivery request" event).

event(delivery_request(store1,coordinates('N',210.000,'W',10.000),
        datime(2010,1,2,0,0,0))).

% 00:01 System matches drivers F,G,H,I,J,K by location and ranking (Driver E
%       is filtered out due to his low rating).
% 00:02 Bid request is sent to drivers F,G,H,I,J,K .

% 00:03 Drivers F,G,H,I,J,K respond with "delivery bid", provide their current
%       location and commit a pick-up time.

event(delivery_bid(1,driverF,coordinates('N',210.000,'W',10.000),
        datime(2010,1,2,0,3,0))).
event(delivery_bid(1,driverG,coordinates('N',210.000,'W',10.000),
        datime(2010,1,2,0,3,0))).
event(delivery_bid(1,driverH,coordinates('N',210.000,'W',10.000),
        datime(2010,1,2,0,3,0))).
```

```
event(delivery_bid(1,driverI,coordinates('N',210.000,'W',10.000),
        datime(2010,1,2,0,3,0))).
event(delivery_bid(1,driverJ,coordinates('N',210.000,'W',10.000),
        datime(2010,1,2,0,3,0))).
event(delivery_bid(1,driverK,coordinates('N',210.000,'W',10.000),
        datime(2010,1,2,0,3,0))).

% 00:03-00:05. Store doesn't perform manual assignment (Alert is sent both to
%       the store and the system manager).

% We leave enough time for the alert to take place
sleep(5).

% 00:06 Driver F is chosen by the store (manual assignment is finally performed
%       after a 3 minute delay). Assignment is sent to driver F. Pick-up time
%       and delivery time is set.

event(assignment(1,store1,coordinates('N',50.000,'W',50.000),
       datime(2010,1,2,0,6,0),driverF,datime(2010,1,2,0,10,0))).

% 00:10 Driver F arrives to the store while exceeding pick-up time by 6 minutes
%       (Pick-Up Alert expected). Pick-up confirmation is set.

sleep(4).
event(pick_up_confirmation(1,driverF,datime(2010, 1, 2, 0, 10, 0))).

% 00:25 F delivers the flowers to the customer while exceeding delivery time by
%       10 minutes (Delivery Alert expected). Delivery confirmation is provided.

sleep(15).
event(delivery_confirmation(1,driverF,datime(2010, 1, 2, 0, 25, 0))).

% 00:31 Rank of F is decreased by one.

% Note: The ranking of the driverA is decreased

%halt.
```

6.2 Aggregates

6.2.1 aggregates_classic_01

```
print_trigger(start_aggr/0).
print_trigger(a/1).
print_trigger(min_comp_event/1).

% min with rules
min_comp_event(100000) <- start_aggr.
min_comp_event(Min) <-
        min_comp_event(MinTemp) seq
        a(X) seq
        prolog(min(X,MinTemp,Min)).
```

```
swipl -g
```

```
2     "open('../results.txt',append,FH),
      ['../../src/etalis.P'],
4      set_etalis_flag(store_fired_events,on),
       compile_event_file('test_01.event'),
6      event(start_aggr),
       event(a(1)),
8      event(a(2)),
       event(a(3)),
10     findall(stored_event(event(min_comp_event(X),T)), stored_event(event(min_comp_event(X),T))↩
           ,List),
       ( List=[ stored_event(event(min_comp_event(100000), [datime(_,_,_,_,_,_,_),datime(_,_,_,_,↩
           _,_,_)])), stored_event(event(min_comp_event(1), [datime(_,_,_,_,_,_,_),datime(_,_,_,_↩
           ,_,_,_)])), stored_event(event(min_comp_event(1), [datime(_,_,_,_,_,_,_),datime(_,_,_,↩
           _,_,_,_)])), stored_event(event(min_comp_event(1), [datime(_,_,_,_,_,_,_),datime(_,_,_↩
           ,_,_,_,_)]))] ->
12         write(FH,'aggregates_classic_01\t\tpassed\n') ;
           write(FH,'aggregates_classic_01\t\tfailed\n') ),halt."
```

6.2.2 aggregates_01

```
1  print_trigger(start_aggr/0).
   print_trigger(a/1).
3  print_trigger(d/1).

5  d(Counter) <- start_aggr seq aggregate(count,a(Y),Counter).
```

```
1  swipl -g
       "open('../results.txt',append,FH),
3      ['../../src/etalis.P'],
        set_etalis_flag(store_fired_events,on),
5       compile_event_file('test_01.event'),
        event(start_aggr),
7       event(a(1)),
        event(a(2)),
9       event(a(0)),
        findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
11      ( List=[stored_event(event(d(0),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
            stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(↩
            event(d(2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(d(3),[↩
            datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])] ->
            write(FH,'aggregates_01\t\t\tpassed\n') ;
13          write(FH,'aggregates_01\t\t\tfailed\n') ),halt."
```

6.2.3 aggregates_02

```
1  print_trigger(start_aggr/0).
   print_trigger(a/1).
3  print_trigger(d/1).

5  d(X) <- start_aggr seq aggregate(sum(Y),a(Y),X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(start_aggr),
     event(a(1)),
     event(a(2)),
     event(a(3)),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     ( List=[stored_event(event(d(0),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(d(3),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(d(6),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'aggregates_02\t\t\tpassed\n') ;
         write(FH,'aggregates_02\t\t\tfailed\n') ),halt."
```

6.2.4 aggregates_03

```
print_trigger(start_aggr/0).
print_trigger(a/1).
print_trigger(d/1).

d(X) <- start_aggr seq aggregate(min(Y),a(Y),X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(start_aggr),
     event(a(1)),
     event(a(2)),
     event(a(3)),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     ( List=[ stored_event(event(d(100000),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'aggregates_03\t\t\tpassed\n') ;
         write(FH,'aggregates_03\t\t\tfailed\n') ),halt."
```

6.2.5 aggregates_04

```
print_trigger(start_aggr/0).
print_trigger(a/1).
print_trigger(d/1).

d(X) <- start_aggr seq aggregate(max(Y),a(Y),X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(start_aggr),
     event(a(1)),
     event(a(2)),
     event(a(3)),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     ( List=[ stored_event(event(d(-100000),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(d(2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(d(3),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'aggregates_04\t\t\tpassed\n') ;
         write(FH,'aggregates_04\t\t\tfailed\n') ),halt."
```

6.3 alarm_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).

% triggers after a second
exceptionAlarm(b(X),1) <- a(X).

% triggers at absolute time: current time Plus one second
exceptionAlarmAbsoluteDatime(c(X),D2) <- a(X) where (
        current_datime(D1), datime_plus_sec(D1,1,D2) ).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)), sleep(2),
     findall(stored_event(event(b(X),T)),stored_event(event(b(X),T)),List),
     findall(stored_event(event(c(X2),T2)),stored_event(event(c(X2),T2)),List2), (
     ( List=[stored_event(event(b(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))], List2=[stored_event(event(c(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))]) ->
         write(FH,'alarm_01\t\t\tpassed\n') ;
         write(FH,'alarm_01\t\t\tfailed\n') ),halt."
```

6.4 and_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
```

```
print_trigger(d/1).

d(X) <- a(X) and b(X).
d(X) <- d(X) and c(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(1)),
     event(c(1)),
     findall(stored_event(event(d(1),T)), stored_event(event(d(1),T)),List), ( List = [↩
         stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),stored_event(↩
         event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'and_01\t\t\t\tpassed\n') ;
         write(FH,'and_01\t\t\t\tfailed') ),halt."
```

6.5 channel_01

```
print_trigger(a/2).
print_trigger(b/2).

fire_event(Agent,Event) :-
    findall(Channel,agent_channel(Agent,Channel),L),
    fire_event_channels(Event,L),
    !.

fire_event_channels(Event,[]):-
    !.
fire_event_channels(Event,[Channel|T]):-
    Event =.. [H|L],
    NewEvent =.. [H,Channel|L],
    event(NewEvent),
    fire_event_channels(Event,T),
    !.

b(Channel,X) <- a(Channel,X).

agent_channel(sensor1,channel1).
agent_channel(sensor1,channel2).
agent_channel(sensor2,channel2).
agent_channel(sensor2,channel3).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Define event types:
%
%event_type(a/1,[number]). % facts
%event_type(b/1,[number]).
%
%check_event_type(X):-
%  X=..[EventName|Args],
%  length(Args,Arity),
%  event_type(EventName/Arity,ArgumentTypes),
```

```
%  check_arg_types(Args,ArgumentTypes).
%
%check_arg_types([],[]):-
%  !.
%check_arg_types([Head|Args],[Type|ArgumentTypes]):-
%  Check=..[Type,Head],
%  call(Check),
%  check_arg_types(Args,ArgumentTypes),
%  !.
%
%Define channels event types:
%
%channel_types(channel1,[a/1,b/1]).
%
%% check if an event matches the types of a channel:
%check_channel_input(Channel,Event):-
%  Event=..[EventName|Args],
%  length(Args,Arity),
%  channel_types(Channel,EventTypes),
%  member(EventName/Arity,EventTypes),
%  check_event_type(Event). % checks if the arguments are ok
%
%The predicate member/2 is the standard one in Prolog. I think its already defined in Etalis utils.P. Maybe its name there is my_member or something like that. Otherwise just search "member Prolog" of Google. Same with length/2.
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'), fire_event(sensor1,a(1)),
     findall(stored_event(event(b(S,X),T)),stored_event(event(b(S,X),T)),List), (List=[stored_event(event(b(channel1,1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(b(channel2,1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'channels_01\t\t\tpassed\n') ;
         write(FH,'channels_01\t\t\tfailed\n') ),halt."
```

6.6 cnot_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).

d(X) <- (a(X) seq b(X)) cnot c(Y).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(1)),
```

```
8       event(a(2)),
        event(c(2)),
10      event(b(2)),
        findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
12      ( List=[stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
            write(FH,'cnot_01\t\t\tpassed\n') ;
14          write(FH,'cnot_01\t\t\tfailed\n') ),halt."
```

6.7 during_01

```
print_trigger(a/1).
2 print_trigger(b/1).
  print_trigger(c/1).
4 print_trigger(d/1).
  print_trigger(e/1).
6
  c(X) <- a(X) seq b(X).
8 d(X) <- a(X) seq b(X).

10 e(X) <- c(X) during d(X).
```

```
swipl -g
2     "open('../results.txt',append,FH),
      ['../../src/etalis.P'],
4      set_etalis_flag(store_fired_events,on),
       compile_event_file('test_01.event'),
6      event(a(1)),
       event(b(1)),
8      findall(stored_event(event(e(X),T)),stored_event(event(e(X),T)),List),
       ( List=[stored_event(event(e(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
10         write(FH,'during_01\t\t\tpassed\n') ;
           write(FH,'during_01\t\t\tfailed\n') ),halt."
```

6.8 dynamic_updates_01

```
1 print_trigger(a/1).
  print_trigger(b/1).
3 print_trigger(c/1).
  print_trigger(d/1).
5 print_trigger(e/1).

7 noop_event <- a(X) seq b(X) do ins_event_rule(d(Y) <- c(Y)).

9 noop_event <- a(X) do ins_static_rule(s1 :- s2).

11 r1 rule: e(X) <- a(X).

13 noop_event <- b(X) do del_event_rule(r1 rule: e(X) <- a(X)).
```

```
% static rules
s2 :- s3.
s3.

% we fire a(1), b(1) and c(2) and we check if event d(2) was detected
%  and if the literal s1 is true
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(1)),
     event(c(1)),
     event(a(1)),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     findall(stored_event(event(e(X),T)),stored_event(event(e(X),T)),List2), ( (List=[↩
         stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))],List2=[↩
         stored_event(event(e(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))],s1) ->
         write(FH,'dynamic_updates_01\t\tpassed\n') ;
         write(FH,'dynamic_updates_01\t\tfailed\n') ),halt."
```

6.9 equals_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).
print_trigger(e/1).

c(X) <- a(X) seq b(X).
d(X) <- a(X) seq b(X).

e(X) <- c(X) equals d(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(1)),
     findall(stored_event(event(e(X),T)),stored_event(event(e(X),T)),List),
     ( List=[stored_event(event(e(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'equals_01\t\t\tpassed\n') ;
         write(FH,'equals_01\t\t\tfailed\n') ),halt."
```

6.10 event_multiply_01

```
print_trigger(a/1).
print_trigger(b/1).

b(Y) <- a(X) event_multiply ( p(X,Y) ).

db( p(1,2) ).
db( p(1,3) ).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     findall(stored_event(event(b(X),T)),stored_event(event(b(X),T)),List),
     ( List=[stored_event(event(b(2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),↩
         stored_event(event(b(3),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'event_multiply_01\t\tpassed\n') ;
         write(FH,'event_multiply_01\t\tfailed\n') ),halt."
```

6.11 finishes_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).
print_trigger(e/1).

c(X) <- a(X) seq b(X).
d(X) <- a(X) seq b(X).

e(X) <- c(X) finishes d(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(1)),
     findall(stored_event(event(e(X),T)),stored_event(event(e(X),T)),List),
     ( List=[stored_event(event(e(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'finishes_01\t\t\tpassed\n') ;
         write(FH,'finishes_01\t\t\tfailed\n') ),halt."
```

6.12 fnot_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(d/1).

d(X) <- b(X) fnot a(Y).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(b(2)),
     event(a(1)),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     ( List=[stored_event(event(d(2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'fnot_01\t\t\t\tpassed\n') ;
         write(FH,'fnot_01\t\t\t\tfailed\n') ),halt."
```

6.13 forall_seq_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(d/1).

d(X) <- a(X) forall_seq b(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(a(1)),
     event(b(1)),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     ( List=[stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
         stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'forall_seq_01\t\t\tpassed\n') ;
         write(FH,'forall_seq_01\t\t\tfailed\n') ),halt."
```

6.14 garbage_collection_general_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(d/1).

```

```
d(X) <- a(X) seq b(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(garbage_control,general),
     set_etalis_flag(garbage_window,1),
     set_etalis_flag(garbage_window_step,1),
     compile_event_file('test_01.event'),
     event(a(1)), sleep(2),
     event(b(1)),
     event(a(2)),
     event(b(2)),
     findall(stored_event(event(d(X),T)), stored_event(event(d(X),T)),List), ( List = [↩
         stored_event(event(d(2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'garbage_collection_general_01\tpassed\n') ;
         write(FH,'garbage_collection_general_01\tfailed\n') ),halt."
```

6.15 garbage_collection_pattern_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).

r1 rule: c(X) <- a(X) seq b(X).
event_rule_property(r1,window,1).
event_rule_property(r1,window_step,1).

r2([property(window,2),property(window_step,1)]) rule:
    d(X) <- a(X) seq b(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)), sleep(2),
     event(b(1)),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     findall(stored_event(event(c(X2),T2)),stored_event(event(c(X2),T2)),List2),
     ( List=[stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))], List2=[]↩
         ->
         write(FH,'garbage_collection_pattern_01\tpassed\n') ;
         write(FH,'garbage_collection_pattern_01\tfailed\n') ), halt."
```

6.16 java_interface_01

```
external_trigger(a/0).
external_trigger(b/0).
external_trigger(c/0).

c <- a seq b.
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(store_fired_events_java,on),
     compile_event_file('test_01.event'), fire_events_java([a],OutputList), fire_events_java([b],OutputList2), nl, nl, write(OutputList), nl, nl, write(OutputList2), nl, halt."
```

6.17 justification_01

```
print_trigger(a/0).
print_trigger(b/0).
print_trigger(c/0).

c <- a seq b.
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(etalis_justification,on),
     compile_event_file('test_01.event'),
     event(a),
     event(b),
     findall(stored_event(event(c,T)),stored_event(event(c,T)),List), ( List = [stored_event(event(c,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'justification_01\t\tpassed\n') ;
         write(FH,'justification_01\t\tfailed\n') ), justify_event(c,[T1,T2],J), write_justification(J), halt."
```

6.18 justification_02

```
print_trigger(a/0).
print_trigger(b/0).
print_trigger(c/0).

c <- a seq b.
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(etalis_justification,on),
     compile_event_file('test_01.event'),
     event(a),
     findall(stored_event(event(c,T)),stored_event(event(c,T)),List), ( List = [] ->
         write(FH,'justification_02\t\tpassed\n') ;
         write(FH,'justification_02\t\tfailed\n') ), nl,nl,write('Justification for NOT c\n'),
             nl, justify_event(c,[T1,T2],J), write_justification(J), halt."
```

6.19 justification_03

```
print_trigger(a/0).
print_trigger(b/0).
print_trigger(c/0).

c <- a seq b.
```

```
rm output.jpg

swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(etalis_justification,on),
     compile_event_file('test_01.event'),
     event(a),
     event(b),
     findall(stored_event(event(c,T)),stored_event(event(c,T)),List), ( List = [stored_event(
         event(c,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'justification_01\t\tpassed\n') ;
         write(FH,'justification_01\t\tfailed\n') ), justify_event(c,[T1,T2],J),
             write_justification(J), write_justification_udraw(justify_event(c,[T1,T2]),J,'
             output.udg'), halt."

uDrawGraph -init ../../lib/remote-uDraw.txt
```

6.20 justification_04

```
print_trigger(a/0).
print_trigger(b/0).
print_trigger(c/0).

c <- a seq b.
```

```
rm output.jpg

swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(etalis_justification,on),
     compile_event_file('test_01.event'),
     event(a),
     findall(stored_event(event(c,T)),stored_event(event(c,T)),List), ( List = [] ->
         write(FH,'justification_02\t\tpassed\n') ;
         write(FH,'justification_02\t\tfailed\n') ), nl,nl,write('Justification for NOT c\n'),
             nl, justify_event(c,[T1,T2],J), write_justification(J), write_justification_udraw(
             justify_event(c,[T1,T2]),J,'output.udg'), halt."

uDrawGraph -init ../../lib/remote-uDraw.txt
```

6.21 meets_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).
print_trigger(e/1).
print_trigger(f/1).

d(X) <- a(X) seq b(X).
e(X) <- b(X) seq c(X).

f(X) <- d(X) meets e(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(1)),
     event(c(1)),
     findall(stored_event(event(f(X),T)),stored_event(event(f(X),T)),List),
     ( List=[stored_event(event(f(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'meets_01\t\t\tpassed\n') ;
         write(FH,'meets_01\t\t\tfailed\n') ),halt."
```

6.22 or_01

```
print_trigger(a/1).
print_trigger(b/1).
```

```
print_trigger(d/1).

d(X) <- a(X) or b(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(2)),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     ( List=[stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),↩
         stored_event(event(d(2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'or_01\t\t\t\tpassed\n') ;
         write(FH,'or_01\t\t\t\tfailed\n') ),halt."
```

6.23 out_of_order_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(d/1).

d(X) <- a(X) seq b(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(out_of_order,on),
     compile_event_file('test_01.event'),
     event(b(1),[2,2]),
     event(a(1),[1,1]),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     ( List=[stored_event(event(d(1),[1,2]))] ->
         write(FH,'out_of_order_01\t\t\tpassed\n') ;
         write(FH,'out_of_order_01\t\t\tfailed\n') ),halt."
```

6.24 par_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).

d(X) <- a(X) seq b(X) par c(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(c(1)),
     event(b(1)),
     event(a(2)),
     event(b(2)),
     event(c(2)),
     findall(stored_event(event(d(X),T)), stored_event(event(d(X),T)),List), ( List = [↩
         stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'par_01\t\t\tpassed\n') ;
         write(FH,'par_01\t\t\tfailed\n') ),halt."
```

6.25 projection_01

```
print_trigger(a/5).
print_trigger(ce/1).

ce(A1) <- a(ID,A1,A2,A3,TS).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1,1,1,1,1)),
     event(a(2,2,2,2,2)),
     event(a(3,3,3,3,3)),
     findall(stored_event(event(ce(A1),T)),stored_event(event(ce(A1),T)),List), ( List = [↩
         stored_event(event(ce(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event↩
         (event(ce(2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(ce(3)↩
         ,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])) ] ->
         write(FH,'projection_01\t\t\tpassed\n') ;
         write(FH,'projection_01\t\t\tfailed\n') ),halt."
```

6.26 projection_join_02

```
print_trigger(a/5).
print_trigger(b/5).
print_trigger(ce/1).

ce(A1) <-
        a(ID,A1,A2,A3,TS) seq
        b(ID,A1,A2,A3,TS).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1,1,1,1,1)),
     event(b(1,1,1,1,1)),
     event(a(2,2,2,2,2)),
     event(b(2,2,2,2,2)),
     event(a(3,3,3,3,3)),
     event(b(3,3,3,3,3)),
     findall(stored_event(event(ce(A1),T)),stored_event(event(ce(A1),T)),List), ( List = [↩
         stored_event(event(ce(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event↩
         (event(ce(2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(event(ce(3)↩
         ,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])) ] ->
         write(FH,'projection_join_02\t\tpassed\n') ;
         write(FH,'projection_join_02\t\tfailed\n') ),halt."
```

6.27 prolog_01

```
print_trigger(a/0).
print_trigger(b/0).
print_trigger(c/0).
print_trigger(d/0).

d <- a seq b seq c.
d <- d seq a.
```

```
sicstus --goal "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(prolog_backend,sicstus),
     compile_event_file('test_01.event'),
     event(a),
     event(b),
     event(c),
     event(a),
     findall(stored_event(event(d,T)), stored_event(event(d,T)),List), ( List = [stored_event(↩
         event(d,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),stored_event(event(d,[datime(_↩
         ,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'prolog_01 sicstus\t\tpassed\n') ;
         write(FH,'prolog_01 sicstus\t\tfailed\n') ),halt."
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a),
     event(b),
     event(c),
     event(a),
```

```
10      findall(stored_event(event(d,T)), stored_event(event(d,T)),List), ( List = [stored_event(↩
            event(d,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),stored_event(event(d,[datime(_↩
            ,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
            write(FH,'prolog_01 swi\t\t\tpassed\n') ;
12          write(FH,'prolog_01 swi\t\t\tfailed\n') ),halt."
```

```
xsb -e "open('../results.txt',append,FH),
2       cd('../../src/'),['etalis.P'], cd('../examples/prolog_01/'),
         set_etalis_flag(store_fired_events,on),
4        set_etalis_flag(prolog_backend,xsb),
         compile_event_file('test_01.event'),
6        event(a),
         event(b),
8        event(c),
         event(a),
10       findall(stored_event(event(d,T)), stored_event(event(d,T)),List), ( List = [stored_event(↩
             event(d,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),stored_event(event(d,[datime(_↩
             ,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
             write(FH,'prolog_01 xsb\t\t\tpassed\n') ;
12           write(FH,'prolog_01 xsb\t\t\tfailed\n') ),halt."
```

```
1   yappl -g
        "open('../results.txt',append,FH),
3       ['../../src/etalis.P'],
         set_etalis_flag(store_fired_events,on),
5        set_etalis_flag(prolog_backend,yap),
         compile_event_file('test_01.event'),
7        event(a),
         event(b),
9        event(c),
         event(a),
11       findall(stored_event(event(d,T)), stored_event(event(d,T)),List), ( List = [stored_event(↩
             event(d,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),stored_event(event(d,[datime(_↩
             ,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
             write(FH,'prolog_01 yap\t\t\tpassed\n') ;
13           write(FH,'prolog_01 yap\t\t\tfailed\n') ),halt."
```

6.28 revision_01

```
1   print_trigger(a/1).
    print_trigger(b/1).
3   print_trigger(d/1).

5   d <- a seq b.
```

```
1   swipl -g
        "open('../results.txt',append,FH),
3       ['../../src/etalis.P'],
         set_etalis_flag(store_fired_events,on),
5        set_etalis_flag(revision_flag,on),
```

```
     compile_event_file('test_01.event'), (event(a),fail;true),
     event(b),
     findall(stored_event(event(d,T)), stored_event(event(d,T)),List), ( List = [] ->
         write(FH,'revision_01\t\t\tpassed\n') ;
         write(FH,'revision_01\t\t\tfailed\n') ), halt."
```

6.29 selection_01

```
print_trigger(a/5).
print_trigger(ce/5).

ce(ID,A1,A2,A3,TS) <-
        a(ID,A1,A2,A3,TS) where
        ID =< 2.
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1,1,1,1,1)),
     event(a(2,2,2,2,2)),
     event(a(3,3,3,3,3)),
     findall(stored_event(event(ce(ID,A1,A2,A3,TS),T)),stored_event(event(ce(ID,A1,A2,A3,TS),T)↩
         ),List), ( List = [stored_event(event(ce(1,1,1,1,1),[datime(_,_,_,_,_,_,_),datime(_,_,↩
         _,_,_,_,_)])),stored_event(event(ce(2,2,2,2,2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,↩
         ,_,_)])) ] ->
         write(FH,'selection_01\t\t\tpassed\n') ;
         write(FH,'selection_01\t\t\tfailed\n') ),halt."
```

6.30 selection_join_02

```
print_trigger(a/5).
print_trigger(b/5).
print_trigger(ce/5).

ce(ID,A1,A2,A3,TS) <-
        a(ID,A1,A2,A3,TS) seq
        b(ID,A1,A2,A3,TS) where
        ID =< 2.
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1,1,1,1,1)),
```

```
    event(b(1,1,1,1,1)),
    event(a(2,2,2,2,2)),
    event(b(2,2,2,2,2)),
    event(a(3,3,3,3,3)),
    event(b(3,3,3,3,3)),
    findall(stored_event(event(ce(ID,A1,A2,A3,TS),T)),stored_event(event(ce(ID,A1,A2,A3,TS),T)↩
        ),List), ( List = [stored_event(event(ce(1,1,1,1,1),[datime(_,_,_,_,_,_,_),datime(_,_,↩
        _,_,_,_,_)])),stored_event(event(ce(2,2,2,2,2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_↩
        ,_,_)])) ] ->
        write(FH,'selection_join_02\t\tpassed\n') ;
        write(FH,'selection_join_02\t\tfailed\n') ),halt."
```

6.31 sequence_01

```
print_trigger(a/0).
print_trigger(b/0).
print_trigger(c/0).

c <- a seq b.
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a),
     event(b),
     findall(stored_event(event(c,T)),stored_event(event(c,T)),List), ( List = [stored_event(↩
         event(c,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'sequence_01\t\t\tpassed\n') ;
         write(FH,'sequence_01\t\t\tfailed\n') ),halt."
```

6.32 sequence_02

```
print_trigger(a/0).
print_trigger(b/0).
print_trigger(c/0).
print_trigger(d/0).

d <- a seq b seq c.
d <- d seq a.
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
```

```
    event(a),
    event(b),
    event(c),
    event(a),
    findall(stored_event(event(d,T)), stored_event(event(d,T)),List), ( List = [stored_event(event(d,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),stored_event(event(d,[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
        write(FH,'sequence_01\t\t\tpassed\n') ;
        write(FH,'sequence_01\t\t\tfailed\n') ),halt."
```

6.33 sequence_03

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).

d(X) <- a(X) seq b(X) seq c(X).
d(X) <- d(X) seq a(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(1)),
     event(c(1)),
     event(a(1)),
     findall(stored_event(event(d(1),T)), stored_event(event(d(1),T)),List), ( List = [stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
        write(FH,'sequence_03\t\t\tpassed\n') ;
        write(FH,'sequence_03\t\t\tfailed\n') ),halt."
```

6.34 sequence_04

```
print_trigger(stock/1).

min_compEvent(1000000) <-
        start_compEvent.
min_compEvent(Min) <-
        (min_compEvent(MinTemp) seq
        stock(1,Price,Volume) )seq
        prolog(min(Price,MinTemp,Min)).

print_trigger(min_compEvent/1).

%final_min(Min) <- min_compEvent(Min) seq stop_compEvent.
```

```
%print_trigger(final_min(_)).

max_compEvent(0) <-
        start_compEvent.
max_compEvent(Max) <-
        max_compEvent(MaxTemp) seq
        stock(1,Price,Volume) seq
        prolog(max(Price,MaxTemp,Max)).

print_trigger(max_compEvent/1).

%final_max(Max) <- max_compEvent(Max) seq stop_compEvent.
%print_trigger(final_max(_)).

buy_compEvent(NewPrice) <-
        ((min_compEvent(MinPrice) and max_compEvent(MaxPrice)) seq
        stock(1,NewPrice,NewVolume) )
        where (
                is(Threshold,MinPrice+(MaxPrice-MinPrice)/10),
                <(NewPrice,Threshold)
        ).

buy_compEvent2(NewPrice) <-
    (min_compEvent(MinPrice) par max_compEvent(MaxPrice)) seq
    stock(1,NewPrice,NewVolume) seq
    prolog(is(Threshold,MinPrice+((MaxPrice-MinPrice)/10))) seq
    prolog(<(NewPrice,Threshold)).

print_trigger(buy_compEvent(_)).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(start_compEvent),
     event(stock(1,10,40)),
     event(stock(1,40,40)),
     event(stock(1,12,40)),
     event(stock(1,12,40)),
     findall(stored_event(event(buy_compEvent(Y),T)), stored_event(event(buy_compEvent(Y),T)),↩
         List), ( List = [stored_event(event(buy_compEvent(10),[datime(_,_,_,_,_,_,1),datime(_,↩
         _,_,_,_,_,_)])),stored_event(event(buy_compEvent(12),[datime(_,_,_,_,_,_,1),datime(_,_↩
         ,_,_,_,_,_)]))] ->
         write(FH,'sequence_04\t\t\tpassed\n') ;
         write(FH,'sequence_04\t\t\tfailed\n') ),halt."
```

6.35 sequence_05

```
print_trigger(ce1/1).
print_trigger(ce2/2).
print_trigger(a/2).

ce2(X,Y3) <- a(X,Y1) seq a(X,Y2) seq a(X,Y3).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1,1)),
     event(a(1,2)),
     event(a(1,3)),
     event(a(1,4)),
     findall(stored_event(event(ce2(X1,X2),T)),stored_event(event(ce2(X1,X2),T)),List), ( List ↩
         = [stored_event(event(ce2(1,3),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),↩
         stored_event(event(ce2(1,4),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'sequence_05\t\t\tpassed\n') ;
         write(FH,'sequence_05\t\t\tfailed\n') ),halt."
```

6.36 sharing_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).

c(X) <- a(X) seq b(X).
d(X) <- a(X) seq b(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(rule_sharing,on),
     set_etalis_flag(rule_sharing_debuging,on),
     set_etalis_flag(output_temporary_files,on),
     compile_event_file('test_01.event'),
     findall(X,event_clause_debuging(eventClause(unlabeled,X,seqf(a(_),b(_)))),List), (List=[↩
         temp_e_1(a(_),b(_))] ->
         write(FH,'sharing_01\t\t\tpassed\n') ;
         write(FH,'sharing_01\t\t\tfailed\n') ),halt."
```

6.37 sharing_02

```
d1(X) <- a(X) seq b(X1) seq c(X2).
d2(X) <- a(X) seq b(X1) seq c(X2).

print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).

print_trigger(d1/1).
print_trigger(d2/1).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(rule_sharing,on),
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(output_temporary_files,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(1)),
     event(c(1)),
     findall(stored_event(event(d1(X),T)),stored_event(event(d1(X),T)),List),
     findall(stored_event(event(d2(X2),T2)),stored_event(event(d2(X2),T2)),List2), ((List=[
         stored_event(event(d1(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))], List2=[
         stored_event(event(d2(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ) ->
         write(FH,'sharing_02\t\t\tpassed\n') ;
         write(FH,'sharing_02\t\t\tfailed\n') ),halt."
```

6.38 star_goal_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(d/1).

d(X) <- a(X) seq b(X) star_times.
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(1)),
     event(b(1)),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     ( List=[stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), 
         stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), stored_event(
         event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'star_goal_01\t\t\tpassed\n') ;
         write(FH,'star_goal_01\t\t\tfailed\n') ),halt."
```

6.39 starts_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).
print_trigger(e/1).
```

```
c(X) <- a(X) seq b(X).
d(X) <- a(X) seq b(X).

e(X) <- c(X) starts d(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(b(1)),
     findall(stored_event(event(e(X),T)),stored_event(event(e(X),T)),List),
     ( List=[stored_event(event(e(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'starts_01\t\t\tpassed\n') ;
         write(FH,'starts_01\t\t\tfailed\n') ),halt."
```

6.40 transitive_closure_01

```
print_trigger(edge_event/2).
print_trigger(reach/2).

reach(X,Y)<- reach(X,Z) seq edge_event(Z,Y).
reach(X,Y)<- edge_event(X,Y).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(event_consumption_policy,recent),
     set_etalis_flag(output_temporary_files,on),
     compile_event_file('test_01.event'),
     event(edge_event(1,2)),
     event(edge_event(2,3)),
     event(edge_event(3,4)),
     findall(stored_event(event(reach(X,Y),T)),stored_event(event(reach(X,Y),T)),List),
     ( List=[stored_event(event(reach(1,2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
         stored_event(event(reach(1,3),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
         stored_event(event(reach(2,3),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
         stored_event(event(reach(2,4),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
         stored_event(event(reach(3,4),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'transitive_closure_01\t\tpassed\n') ;
         write(FH,'transitive_closure_01\t\tfailed\n') ),halt."
```

6.41 transitive_closure_02

```
print_trigger(edge_event/2).
```

```
print_trigger(reach/2).

reach(X,Y)<- reach(X,Z) seq edge_event(Z,Y).
reach(X,Y)<- edge_event(X,Y).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     set_etalis_flag(event_consumption_policy,unrestricted),
     compile_event_file('test_01.event'),
     event(edge_event(1,2)),
     event(edge_event(2,3)),
     event(edge_event(3,4)),
     findall(stored_event(event(reach(X,Y),T)),stored_event(event(reach(X,Y),T)),List),
     ( List=[stored_event(event(reach(1,2),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
         stored_event(event(reach(1,3),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
         stored_event(event(reach(2,3),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
         stored_event(event(reach(1,4),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
         stored_event(event(reach(2,4),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])), ↩
         stored_event(event(reach(3,4),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'transitive_closure_02\t\tpassed\n') ;
         write(FH,'transitive_closure_02\t\tfailed\n') ),halt."
```

6.42 where_01

```
print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).

d(X) <- a(X) where (X>0).

d(Y) <- a(X) seq b(Y) where (Y>X).

d(Z) <- a(X) seq b(Y) seq c(Z) where (Y>X,Z>Y).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)),
     event(a(0)),
     event(b(1)),
     event(c(2)),
     findall(stored_event(event(d(X),T)), stored_event(event(d(X),T)),List), ( List = [↩
         stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),stored_event(↩
         event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)])),stored_event(event(d(2),[↩
         datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))] ->
         write(FH,'where_01\t\t\tpassed\n') ;
         write(FH,'where_01\t\t\tfailed\n') ),halt."
```

6.43 windows_01

```
% note: see examples/garbage_collection_pattern_01 for time windows + garbage
%       collection together (it uses another opperand, namely window)

print_trigger(a/1).
print_trigger(b/1).
print_trigger(c/1).
print_trigger(d/1).

r1 rule: c(X) <- a(X) seq b(X).
event_rule_property(r1,event_rule_window,1).

r2([property(event_rule_window,3)]) rule: d(X) <- a(X) seq b(X).
```

```
swipl -g
    "open('../results.txt',append,FH),
    ['../../src/etalis.P'],
     set_etalis_flag(store_fired_events,on),
     compile_event_file('test_01.event'),
     event(a(1)), sleep(2),
     event(b(1)),
     findall(stored_event(event(d(X),T)),stored_event(event(d(X),T)),List),
     findall(stored_event(event(c(X2),T2)),stored_event(event(c(X2),T2)),List2), ((List=[↩
         stored_event(event(d(1),[datime(_,_,_,_,_,_,_),datime(_,_,_,_,_,_,_)]))], List2=[]) ->
         write(FH,'windows_01\t\t\tpassed\n') ;
         write(FH,'windows_01\t\t\tfailed\n') ), halt."
```

Chapter 7

Event Processing SPARQL (EP-SPARQL)

To enable ETALIS system to handle real time Semantic Web applications we have developed Event Processing SPARQL (EP-SPARQL) language. This extension enables a user to specify event patterns in a SPARQL-like language. Event streams are expected to be in an RDF format (i.e., RDF streaming triples additionally accompanied with timestamps). The background (contextual) knowledge can be specified as an RDFS ontology.

Syntactically, we defined EP-SPARQL to be SPARQL extended by the binary operators SEQ, EQUALS, OPTIONALSEQ, and EQUALSOPTIONAL used to combine graph patterns in the same way as UNION and OPTIONAL in pure SPARQL. Intuitively, all those operators act like a (left, right or full) join, but they do so in a selective way depending on how the constituents are temporally interrelated, as indicated by their naming: P_1 SEQ P_2 joins P_1 and P_2 only if P_2 occurs strictly after P_1, whereas P_1 EQUALS P_2 performs the join if P_1 and P_2 are exactly simultaneous. OPTIONALSEQ and EQUALSOPTIONAL are temporal-sensitive variants of OPTIONAL.

Moreover, we added the function `getDURATION()` to be used inside filter expressions. This function yields a literal of type `xsd:duration` giving the length of the time interval associated to the graph pattern the FILTER condition is placed in. Likewise, we added functions `getSTARTTIME()` and `getENDTIME()` to retrieve the time stamps (of type `xsd:dateTime`) of the start and end of the currently described interval.

7.0.1 Examples with EP-SPARQL

We provide a few examples to give some intuition on EP-SPARQL operators supported by ETALIS system. The following EP-SPARQL query is supposed to search for companies whose stock price has decreased by over 30% and subsequently risen by more than 5% within a time frame of 30 days.

```
SELECT ?company
WHERE
       { ?company hasStockPrice ?price1 }
   SEQ { ?company hasStockPrice ?price2 }
   SEQ { ?company hasStockPrice ?price3 }
FILTER ( ?price2 < ?price1 * 0.7 &&             (1)
         ?price3 > ?price1 * 1.05 &&
         getDURATION() < "P30D"^^xsd:duration)
```

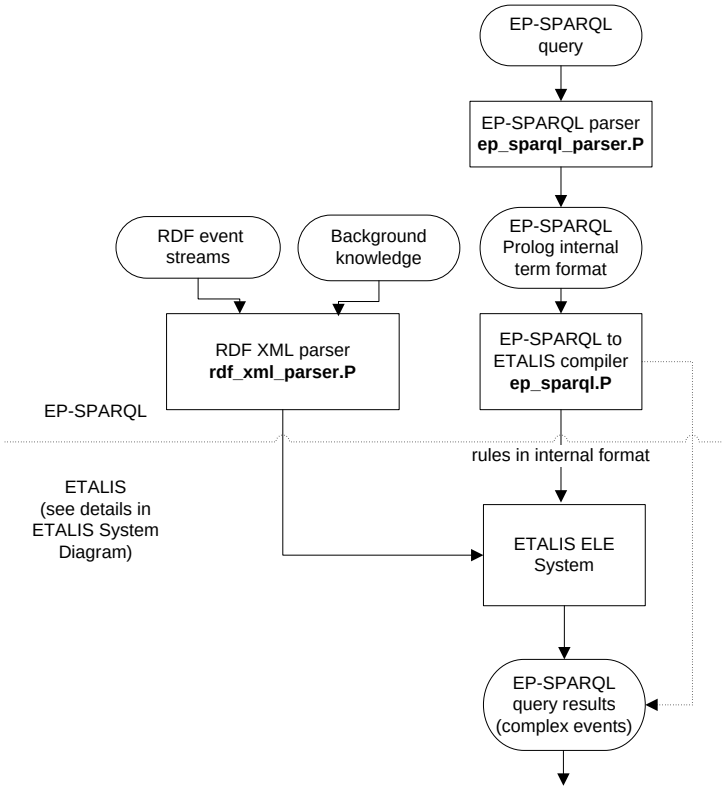

Figure 7.1: System Diagram: EP-SPARQL

The next EP-SPARQL query will identify companies with a more than 50% stock price drop and – in case some rating agency previously downrated this company, this rating agency will be indicated as well.

```
SELECT ?company ?ratingagency
WHERE
       { ?company downratedby ?ratingagency}
OPTIONALSEQ                                        (2)
       { { ?company hasStockPrice ?price1 }
   SEQ { ?company hasStockPrice ?price2 } }
FILTER ( ?price2 < ?price1 * 0.5)
```

It is worth mentioning that – just like for pure SPARQL – negation (i.e., requiring the *absence* of some triple pattern instead of its *presence*) is not an explicit part of the formalism, but can be expressed via OPTIONAL and FILTER. For instance, the following query asks for companies having a larger than 50% stock price increase in less than 15 days without having acquired another company during that period.

```
SELECT ?company
WHERE
       { ?company hasStockprice ?price1 }
   SEQ { { ?company hasAcquired ?othercompany }
OPTIONALSEQ
       { ?company hasStockPrice ?price2 } }
```

```
FILTER ( ?price2 < ?price1 * 1.5 &&                (3)
         !BOUND(?othercompany) &&
         getDURATION() < "P15D"^^xsd:duration )
```

Moreover, we allow for recursion by employing CONSTRUCT queries, conceiving them as a kind of production rule. Thereby, the result graph of such a query is assumed to be added to the RDF stream. For instance, the following statement gathers "temporally distributed" rating information to create a triple indicating an event of being downrated, which in turn can be used in other CONSTRUCT or SELECT queries.

```
CONSTRUCT ?company downratedby ?ratingagency
WHERE
       { ?rating1 rater ?ratingagency ;
         rated ?company ; score ?score1 }
   SEQ { ?rating2 rater ?ratingagency ;           (4)
         rated ?company ; score ?score2 }
FILTER ( ?score2 < ?score1 )
```

Finally, the forthcoming extended SPARQL standard[1] featuring *subqueries* and *expressions* allows for as complex mechanisms as aggregation over *sliding windows*. As an example we present a query monitoring the average stock price of a company ACME Inc. over the last 10 days. First, we use a construct rule that aggregates counts and sums of stock prices within the given time frame and feeds this information back into the stream. Thereby, the EQUALSOPTIONAL and filter part make sure that no price signal is left out.

```
CONSTRUCT  _:aaa   :hasCount ?count .
           _:aaa   :hasSum   ?sum .
{ SELECT   ?count AS ?prevcount + 1
           ?sum   AS ?prevsum + ?price
 WHERE {{ ?point :hasCount ?prevcount .
             ?point :hasSum  ?prevsum  . }
   SEQ { :ACME  :hasStockPrice  ?price . }}
   EQUALSOPTIONAL                                  (5)
       {{ ?point :hasCount ?prevcount .
          ?point :hasSum  ?prevsum  . }
   SEQ { :ACME  :hasStockPrice  ?inbetween . }
   SEQ { :ACME  :hasStockPrice  ?price . }}
FILTER ( !BOUND(?inbetween) &&
         getDURATION() < "P10D"^^xsd:duration )}
```

7.0.2 Internals of EP-SPARQL Implementation

EP-SPARQL is implemented as an extension to ETALIS system, described in Section **??**. A system diagram of our EP-SPARQL engine is shown in Figure 7.1.

A user is expected to write EP-SPARQL queries and to deploy them in the engine. These queries act similarly as *continuous* queries in Database Stream Management Systems (DSMS), i.e., once registered the queries are continuously evaluated with respect to streaming data. In our implementation, the engine *incrementally* matches incoming data (events) and produce complex events that satisfy queries as soon as they occur (see Section **??**).

Since event streams and the background knowledge are represented in the RDF format, we use an RDF/XML parser to convert inputs into internal ETALIS format (see Figure 7.1). For event streams the conversion is applied on-the-fly. It is a straight forward mapping, and typically does not case a significant overhead at run time. The background knowledge (RDFS ontologies) can be converted in Prolog representation at design time. Similarly, we have also implemented a parser for EP-SPARQL syntax and a compiler which produces EDBC rules out of EP-SPARQL expressions. All three inputs (EP-SPARQL queries, event streams and the domain ontology) are then fed into ETALIS system where the processing, as described in Section **??**, takes place.

[1] http://www.w3.org/TR/2009/WD-sparql-features-20090702/

Bibliography

[1] J. F. Allen. Maintaining knowledge about temporal intervals. In *Communications of the ACM 26, 11, 832-843*, 1983.

[2] D. Anicic, P. Fodor, R. Sthmer, and N. Stojanovic. An approach for data-driven and logic-based complex event processing, abstract paper. In *DEBS*, 2009.

[3] D. Anicic, P. Fodor, R. Sthmer, and N. Stojanovic. Computing complex events in an event-driven and logic-based approach. system demo. In *DEBS*, 2009.

[4] D. Anicic, P. Fodor, R. Sthmer, and N. Stojanovic. Efficient logic-based complex event processing and reactivity handling. In *Technical Report*, 2009. `http://code.google.com/p/etalis/wiki/TechnicalReport`.

[5] D. Anicic, P. Fodor, R. Sthmer, and N. Stojanovic. Event-driven approach for logic-based complex event processing. In *CSE*, 2009.

[6] P. Fodor and D. Anicic. The Fast Flower Delivery Use Case in ETALIS. `http://code.google.com/p/etalis/wiki/\Fast_Flower_Delivery_Use_Case`.

[7] M. Li, M. Liu, L. Ding, E. A. Rundensteiner, and M. Mani. Event stream processing with out-of-order data arrival. In *ICDCSW*, 2007.

[8] The ETALIS Team. The ETALIS Web site. `http://code.google.com/p/etalis`.

Index